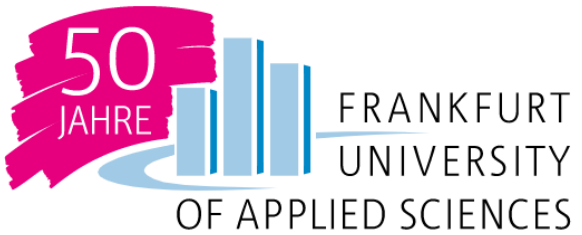

Frankfurt University of Applied Sciences

Cloud Computing
WS 20/21

OpenFaaS Installation Guide on Single-Node and Multi-Node Kubernetes Cluster

Kshitij Yelpale (1322509)
Sohail Dua (1322512)
Safir Mohammad Shaikh (1322554)
Karishma (1322486)

Guidance:
Prof. Dr. Christian Baun

Table of Contents

1. Introduction

One of the emerged topics in industries currently is the transition from Monolith to Micro-services to Serverless architecture. Serverless computing is the cutting-edge technology now-a-days that improves the traditional client-server architecture by optimizing load on resources. It describes the concept of developing and running applications without having to worry about the servers. However, this does not illustrate that there are no servers at all but, the overhead of creating and mananging servers is overcome. Developers only have to create new pieces of features as functinos on a cloud service. Due to its light weight architecture, it allows efficient scaling. In this approach, almost all operating concerns are abstracted away from developers. The platform then takes care of function execution, storage, container infrastructure, networking and fault tolerance. Additionally, the serverless platform takes care of scaling the functions according to the actual demand. Currently, all major cloud service providers offer solutions for serverless computing, namely, Amazon Web Services (AWS) Lambda, Azure Functions Serverless Compute (Microsoft), IBM Cloud Functions and Cloud Functions (Google) as depicted in figure 1. Also, this architecture is built by removing database from server and running individual functions on demand and hence it is also referred to as function-as-a-service (FaaS). Primarily, it overcomes 2 major drawbacks of client-server architecture:

- **Single Point of Failure:** Unlike the traditional approach, like in micro-services architecture, one does not need to be completely dependent on the server and worry about the failure of servers.
- **Always Paying:** And, one only needs to pay for the amount of time the services are used.

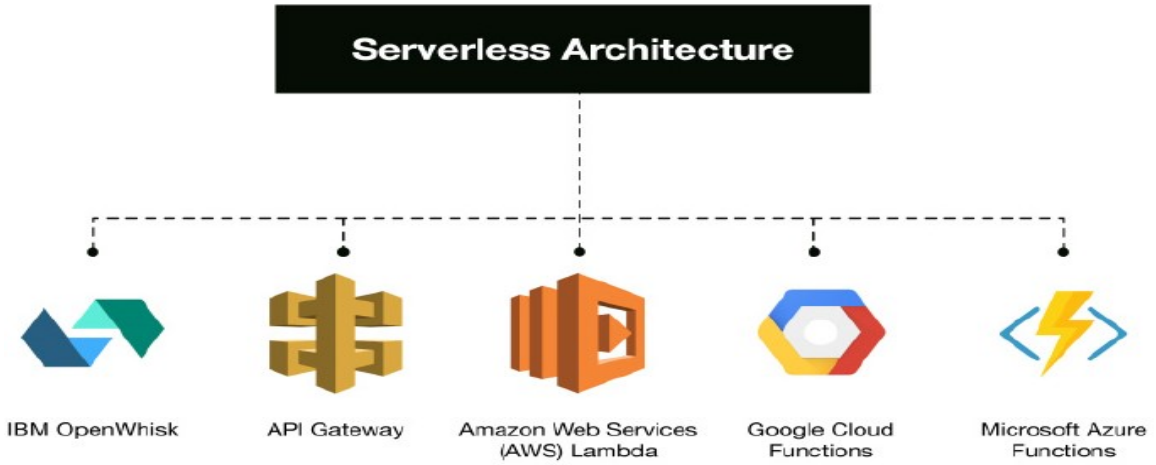

Fig. 1. Serverless Providers [1]

OpenFaaS platform is a promising solution to carry the power of serverless computing on-premises. OpenFaaS is a serverless open-source platform for Docker and Kubernetes. It

is distributed under the MIT license. These frameworks offer greater flexibility (to deploy software, customize the system, etc.) and thus prevent the lock-in of vendors. Open-source systems, for example, can be implemented on both edge/fog devices and the public cloud for distributed data analytics. In this regard, a serverless architecture should be simple to set up, configure and maintain in this respect; it should also provide some guarantees of performance. It is rapidly becoming a famous paradigm because of efficient code maintenance, fewer prices for hosting, and the peace of mind due to running the functions on managed infrastructure.

2. Architecture

The OpenFaaS cloud architecture consists of following modules [2]:

- CICD Pipeline
- Dashboard
- Container Builder
- Authentication Service
- Edge Router

The CICD pipeline is made up of many OpenFaaS functions and the dashboard is served through OpenFaaS. An edge-router maps the user domains to endpoints. Kubernetes is the recommended platform for deploying OpenFaaS, be it local environment, self-hosted cluster, or with a managed service. The API gateway, the feature watchdog, and an example of Prometheus are the components of OpenFaaS. The Gateway can be availed through REST API, via the UI, or through the CLI. The OpenFaaS CLI is used for the creation and implementation of OpenFaaS functions. The developer must supply only the function and the handler, and the CLI manages the packing of the function into a Docker container.

Faas-netes is the widely used orchestration service for OpenFaaS. By communicating with the container orchestrator plugin, an API gateway provides an external interface to the features, collects metrics, and manages to scale. By adjusting the service replica count in the Docker Swarm or Kubernetes, API Gateway can scale functions according to demand.

The container also includes a watchdog feature, i.e. a web server that functions inside the system as an entry point for function calls. By adding Feature Watchdog, we can turn any Docker image into a serverless function (a tiny Golang HTTP server). It is the entry point that enables the forwarding of HTTP requests to the target process. Alert manager reads the metrics collected by the Prometheus and informs the gateway to scale the replica of the functions if needed. The rules for sending alerts can be defined in the configuration file. The conceptual diagram of OpenFaaS by Alex Ellis is illustrated in figure 2.

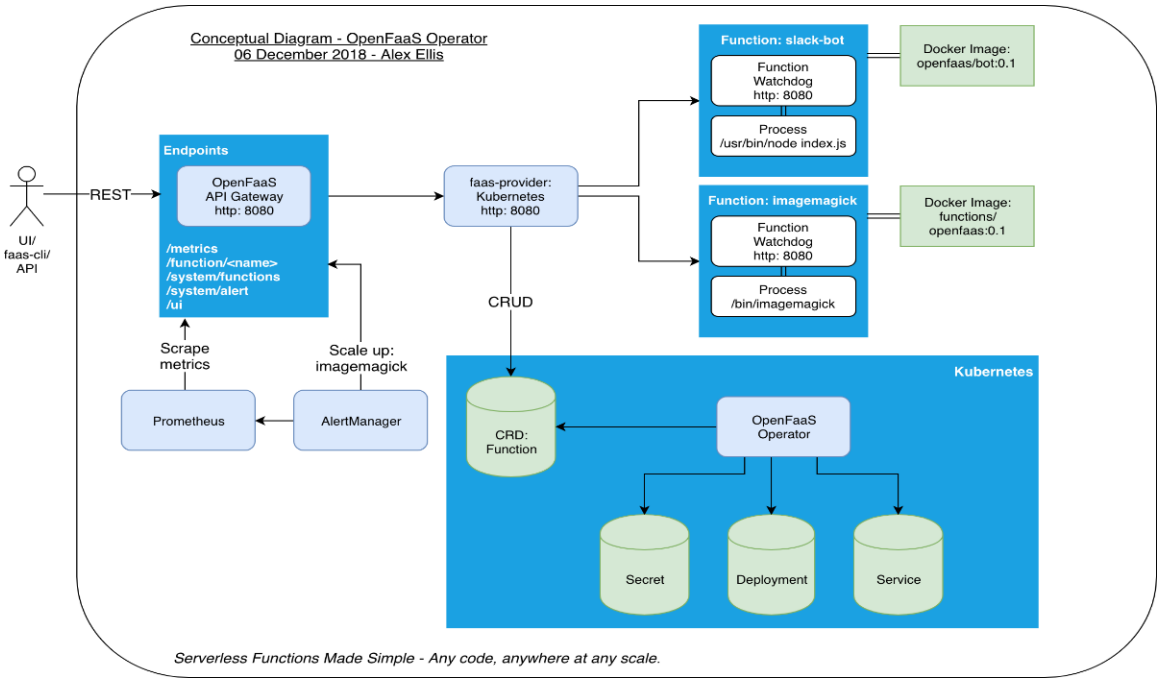

Fig. 2. OpenFaaS Architecture - Alex Ellis
[3]

3. Installation

OpenFaaS may be executed in several contexts. We have implemented OpenFaaS on

- Single Node Kubernetes Cluster - **Minikube**
- Multi Node **Custom** Kubernetes Cluster - 3 EC2 instances on AWS
- Multi Node Kubernetes Cluster using **Managed Service (AWS EKS)**

Note: All the commands mentioned in this report can be executed by a normal user, however, we have included **sudo** in the commands which need to be executed by the root user. Moreover, the **'#'** symbol represents a comment in the command block.

3.1 Single-Node Kubernetes Cluster

Kubernetes coordinates a highly available cluster of computers linked to work as a single unit. Kubernetes' abstractions allow us to deploy containerized applications to a cluster without specifically tying them to individual machines. Kubernetes more efficiently automates the distribution and scheduling of application containers across a cluster. Kubernetes is a framework for open-source development and ready for production. A Kubernetes cluster consists of two types of resources:

- The cluster is coordinated by the master node.
- The worker nodes runs application.

Kubernetes can be used on both physical and virtual machines. Minikube is the component of Kubernetes which can be used for development on local systems. It is a lightweight implementation of Kubernetes that creates a VM on our local machine and uses only one node in its simple Cluster and it serves a master role. Minikube CLI provides basic bootstrapping operations, including start, stop, status, and delete for working with your cluster.

3.1.1 Minikube

Please follow below instructions in order to create a cluster using Minikube on your system [4].

- For best practices, always update your system packages to the latest release. Run following commands to achieve the same:

Update System Packages

```
sudo apt-get update
sudo apt-get install apt-transport-https
sudo apt-get upgrade
```

- Now, download Minikube, assign execute permissions to the downloaded package and add the directory to bin folder. Verify the instllation by executing version command. This step can be viewed in figure 3.

Download Minikube

```
wget  https://storage.googleapis.com/minikube/releases/
   latest/minikube-linux-amd64
chmod +x minikube-linux-amd64
sudo mv minikube-linux-amd64 /usr/local/bin/minikube
minikube version
```

Fig. 3. Get Minikube

- Next, Install **kubectl**. To do the same, please execute the steps given in section 'Common Steps for Kubernetes'.
- Now that components are installed, we can start Minikube to create the single node cluster. Also, check the cluster created with one running master node. This step is shown in figure 4.

Start Minikube

```
minikube start
kubectl get nodes
```

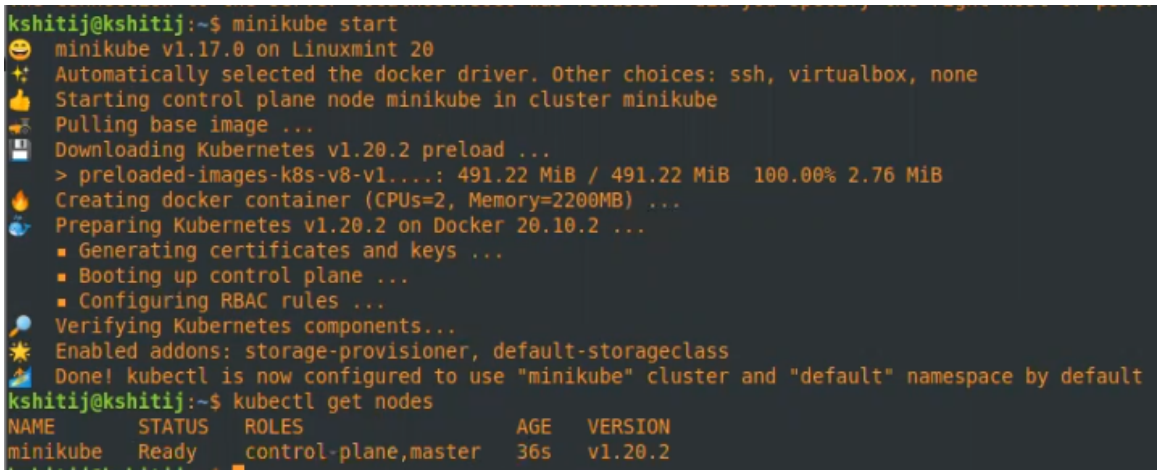

Fig. 4. Start Minikube

- Since the cluster is created, lets move on to OpenFaaS. Please jump to section 'Common Steps for OpenFaaS' and continue execution from there.

3.1.2 Common Steps for Kubernetes

- We need kubectl which is a command line tool used to deploy and manage applications on Kubernetes.

Install kubectl

```
curl -LO https://storage.googleapis.com/kubernetes-release/
    release/`curl -s https://storage.googleapis.com/
    kubernetes-release/release/stable.txt`/bin/linux/amd64/
    kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

3.1.3 Common Steps for OpenFaaS

This section focuses on the commons steps which are required to run before OpenFaaS [5].

- Install OpenFaaS CLI on the top of the Kubernetes cluster. It is a client of OpenFaaS which is used to communicate through the command line. This step is depicted in figure 5.

Install faas-cli

```
curl -sL cli.openfaas.com | sudo sh
```

Fig. 5. Download OpenFaaS

- Helm is a repository tool which is used to download OpenFaaS deployments. The output to the following commands should be similar to how it is in figure 6.

Install Helm

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com
   /helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
helm version
```

```
kshitij@kshitij:~$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/
get-helm-3
kshitij@kshitij:~$ chmod 700 get_helm.sh
kshitij@kshitij:~$ ./get_helm.sh
Helm v3.5.0 is available. Changing from version v3.4.1.
Downloading https://get.helm.sh/helm-v3.5.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
kshitij@kshitij:~$ helm version
version.BuildInfo{Version:"v3.5.0", GitCommit:"32c22239423b3b4ba6706d450bd044baffdcf9e6", GitTreeState:"
clean", GoVersion:"go1.15.6"}
```

Fig. 6. Get Helm

- Let's separate all the deployments (container abstraction) of OpenFaaS in a separate namespace as depicted in figure 7.

Create OpenFaaS Namespaces

```
kubectl apply -https://raw.githubusercontent.com/openfaas/
   faas-netes/master/namespaces.yml
```

```
kshitij@kshitij:~$ kubectl get namespace
NAME              STATUS   AGE
default           Active   2m3s
kube-node-lease   Active   2m4s
kube-public       Active   2m4s
kube-system       Active   2m4s
kshitij@kshitij:~$ kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-netes/master/namespa
ces.yml
namespace/openfaas created
namespace/openfaas-fn created
kshitij@kshitij:~$ kubectl get namespace
NAME              STATUS   AGE
default           Active   2m17s
kube-node-lease   Active   2m18s
kube-public       Active   2m18s
kube-system       Active   2m18s
openfaas          Active   3s
openfaas-fn       Active   3s
```

Fig. 7. Apply Namespaces

- Add OpenFaaS repository to Helm and update the charts. This step is illustrated in figure 8.

Add OpenFaaS to Helm repository and update Helm charts

```
helm repo add openfaas https://openfaas.github.io/faas-
    netes/
helm repo update
```

```
kshitij@kshitij:~$ helm repo add openfaas https://openfaas.github.io/faas-netes/
"openfaas" already exists with the same configuration, skipping
kshitij@kshitij:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "openfaas" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. *Happy Helming!*
```

Fig. 8. Happy Helming!!!

- For the authentication of OpenFaas, we need to have a password which can be generated randomly.

Generate and display a random password for basic authentication

```
export PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d'
    ' -f1)
echo $PASSWORD
```

- Now, Basic auth needs to be created using above generated password.

Creation of basic-auth for OpenFaaS

```
kubectl -n openfaas create secret generic basic-auth --from
    -literal=basic-auth-user=admin --from-literal=basic-auth
    -password="$PASSWORD"
```

- List all available namespaces.

Display all Namespaces

```
kubectl get namespaces
```

- Let's switch to the openfaas namespace so that there will be no need to mention namespace in every command. To set the required namespace, first, get the kubernetes context by following command.

Get the Context

```
kubectl config current-context
```

- Now, set the namespace using above fetched context. Since above command is executed in the minikube installation process, minikube is mentioned here as a context name but this name will be different in other cases. This step is depicted along with the previous step in figure 9.

Set the namespace

```
kubectl config set-context minikube --namespace openfaas
```

```
kshitij@kshitij:~$ kubectl config current-context
minikube
kshitij@kshitij:~$ kubectl config set-context minikube --namespace openfaas
Context "minikube" modified.
kshitij@kshitij:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
alertmanager-644957df4b-28nv9       1/1     Running   0          2m29s
basic-auth-plugin-bc899c574-5rghl   1/1     Running   0          2m29s
gateway-8cd78dcdb-nwv7x             2/2     Running   0          2m29s
nats-7d86c64647-vl9fp               1/1     Running   0          2m29s
prometheus-77f65d99b5-l2b9r         1/1     Running   0          2m29s
queue-worker-77f4446d48-2b84n       1/1     Running   1          2m29s
```

Fig. 9. Set Context

3.1.4 OpenFaaS

Once the steps from section 'Common Steps for OpenFaaS' are covered, proceed with the following steps which are unique to Minikube [5]:

- Now, download the OpenFaaS deployments with below commands as shown in figure 10.

Download and Create OpenFaaS Deployments

```
helm upgrade openfaas --install openfaas/openfaas --
    namespace openfaas --set functionNamespace=openfaas-fn
    --set basic_auth=true
```

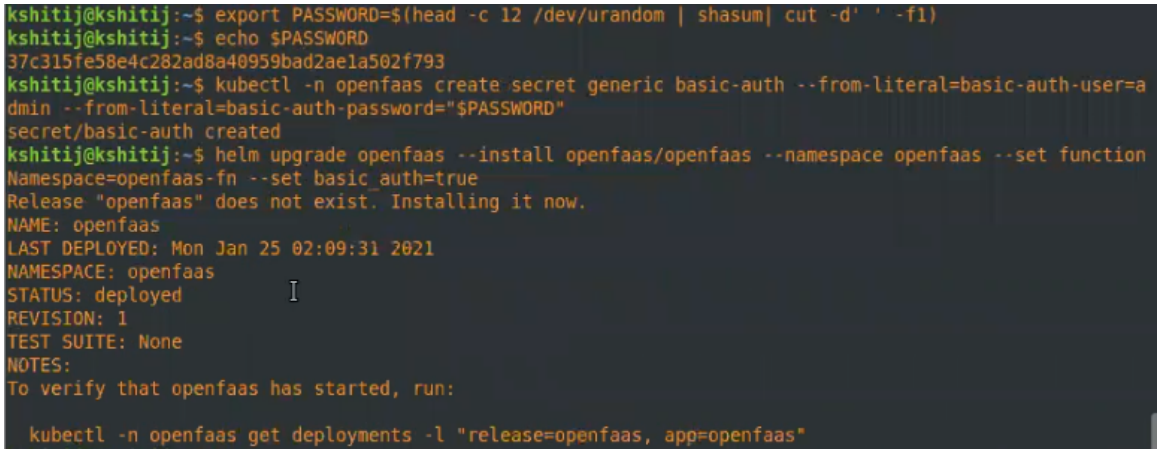

Fig. 10. Get OpenFaaS Deployments

- Get the URL to access OpenFaaS and communicate with commands and set it as an environment variable. This step is displayed in figure 11.

Set OpenFaaS URL

```
export OPENFAAS_URL=$(minikube ip):31112
echo $OPENFAAS_URL
```

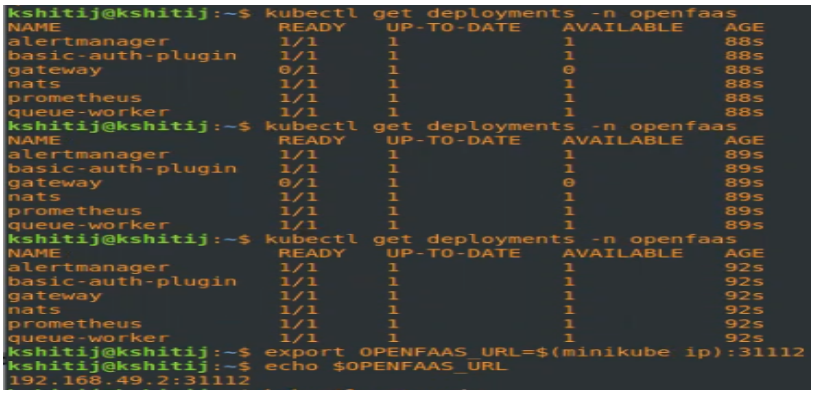

Fig. 11. Get OpenFaaS URL

- List down pods, deployments and services to track the status of running instances.

Get the status of Containers

```
kubectl get pods, deployment, svc
```

- Please wait until all the containers are ready and once all the Pods are started, you can login using the CLI as shown in figure 13. Moreover, you can also access the OpenFaaS portal in your browser directly using the OpenFaaS URL as depicted in figure 12. Meanwhile, you can verify the status by executing the above command continuously.

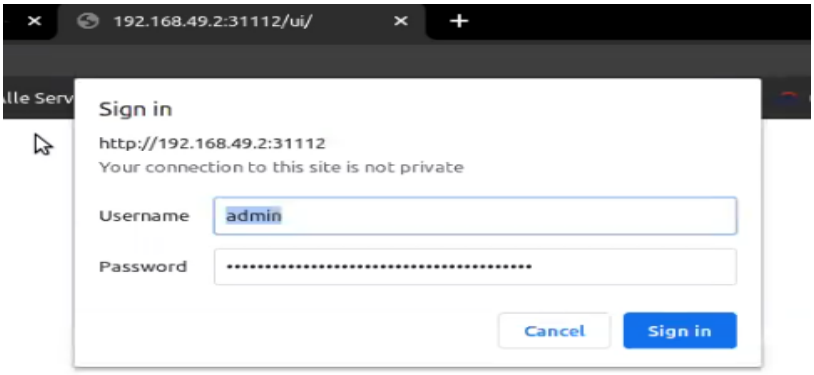

Fig. 12. Access the URL and Authenticate

- All set! Now login to OpenFaas and explore!

Login to OpenFaaS through CLI

```
echo -n $PASSWORD | faas-cli login -g http://$OPENFAAS_URL
   -u admin --password-stdin
```

```
kshitij@kshitij:~$ echo -n $PASSWORD | faas-cli login -g http://$OPENFAAS_URL -u 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 c
ertificates.
credentials saved for admin http://192.168.49.2:31112
```

Fig. 13. Login through CLI

3.2 Multi-Node Kubernetes Cluster

We can use OpenFaas on multi node cluster in several ways and we have implemented two of the many possibilities.

3.2.1 Custom Made Cluster

We can create a custom cluster on our own with multiple nodes, one of them serving as a Master node and other nodes serving as Worker nodes. There are several steps involved in creating such type of cluster. We have used AWS for implementing the same. AWS (Amazon Web Services) provides on-demand cloud computing services on a metered pay-as-you-go basis. AWS provides a free-tier account for new users with limited services free of cost for one year. AWS uses Public Key Cryptography to encrypt and decrypt Login Information where AWS stores Public Key and User stores Private Key.

- Create an AWS Account.
- Create a user with Programmatic access
  - Click on **Add user**
  - Enter user name and select **Programmatic access.** Click on Next.
  - Click on Create Group. Assign Group name. Select the policy as **AdministratorAccess.** Click on **Create group**.
  - Go ahead and click on **Download .csv** so that these credentials can be used to configure your account on AWS CLI as depicted in figure 14.

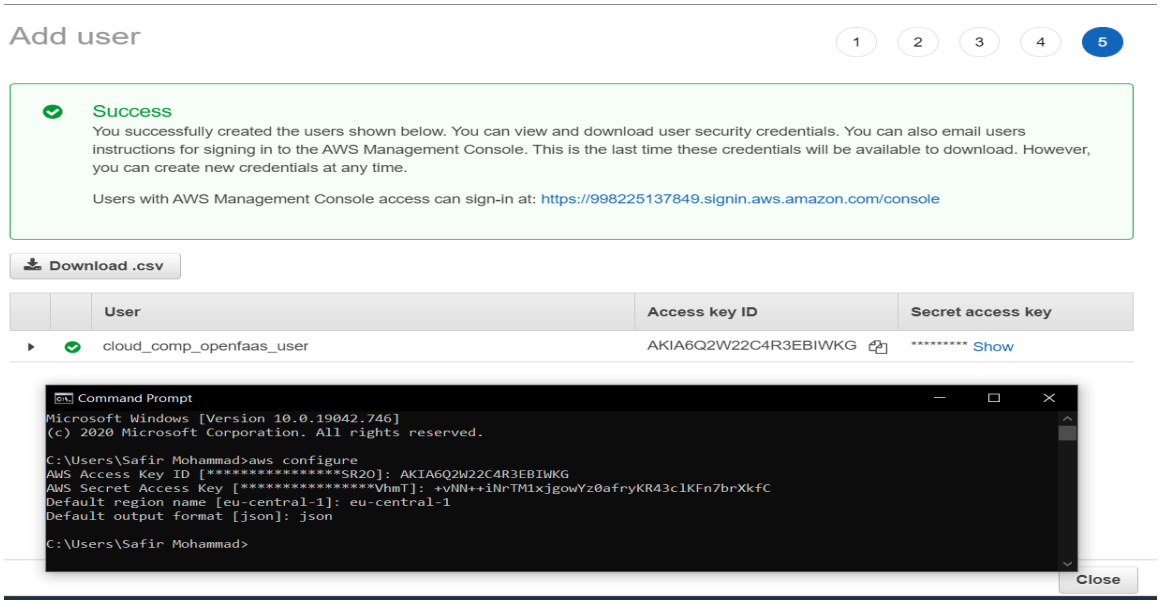

Fig. 14. Configure User in CLI

- Install AWS CLI

Install AWS CLI

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- Configure AWS CLI

Configure AWS CLI

```
aws configure
```

- AWS VPC (Virtual Private Cloud) is a virtual network in public cloud which can be configured according to the requirements. We have following architecture:
  - 1 VPC with IP Range 10.0.0.0/16
  - 1 Public Subnet within the VPC with IP Range 10.0.1.0/24
  - 3 EC2 Instances (Nodes) in the Public Subnet
  - All the resources in Public Subnet connected to Internet Gateway
  - Rules in Route Table:

Destinition	Target	Status	Propagated
10.0.0.0/16	local	active	No
0.0.0.0/0	IGW-ID		No

  - Rules in Security Group - Inbound Rules:

Type	Protocol	Port Range	Source	Description
All Traffic	All	All	Custom	
All Traffic	All	All	My IP/[Anywhere]	

  - Rules in NACL - Inbound Rules:

Rule	Type	Protocol	Port Range	Source	Allow/Deny
100	All Traffic	All	All	0.0.0.0/0	Allow

Let's build above specified architecture now:

AWS VPC

```
# Create VPC
aws ec2 create-vpc --cidr-block 10.0.0.0/16
# Use the generated VPC_ID in upcoming commands
aws ec2 create-tags --resources VPC_ID --tags Key=Name,
    Value=my-cloud-vpc
# Create Internet Gateway and Attach it to VPC
aws ec2 create-internet-gateway
# Use the generated IGW_ID in upcoming commands
aws ec2 create-tags --resources IGW_ID --tags Key=Name,
    Value=my-cloud-igw
aws ec2 attach-internet-gateway --internet-gateway-id
    IGW_ID --vpc-id VPC_ID
# Create Subnet
aws ec2 create-subnet --vpc-id VPC_ID --cidr-block
    10.0.1.0/24
# Use the generated SUBNET_ID in upcoming commands
aws ec2 create-tags --resources SUBNET_ID --tags Key=Name,
    Value=my-cloud-pub-subnet
# Fetch the Route Table ID (From AWS Console) and Add the
    rule
aws ec2 create-tags --resources RTB_ID --tags Key=Name,
    Value=my-cloud-pub-rt
aws ec2 create-route --route-table-id RTB_ID --destination-
    cidr-block 0.0.0.0/0 --gateway-id IGW_ID
# Create Subnet Association with the Route Table
aws ec2 associate-route-table --route-table-id RTB_ID --
    subnet-id SUBNET_ID
# Identify the Security Group corresponding to the
    SUBNET_ID and Add tags
aws ec2 create-tags --resources SEC_GRP_ID --tags Key=Name,
    Value=my-cloud-sec-grp
# Identify the NACL corresponding to the SUBNET_ID and Add
    tags
aws ec2 create-tags --resources NACL_ID --tags Key=Name,
    Value=my-cloud-nacl
```

- AWS EC2 (Elastic Compute Cloud) is a web service that allows you to launch VMs in the AWS cloud with inbuilt easy web-scale computing. One can obtain and launch a single VM or 1000s of VMs in minutes. While creating a VM, we have to select an AMI (Amazon Machine Image) which is a template containing S/W configurations including OS, application server, and applications required to launch the instance. And AWS provides many kinds of AMIs. Some of the most commonly used AMIs include Amazon Linux AMI, Redhat Enterprise, Ubuntu, and Windows. Now, let's create 3 EC2 instances of Ubuntu AMI.

- Go to AWS Management Console through your favourite browser. Click on **Services**. Select **EC2** under **Compute**.
- Click on **Instances**. Select **Launch Instances**.
- Select AMI as **Ubuntu Server 20.04 LTS (HVM), SSD Volume Type**.
- Choose an Instance Type as **t2.micro**.
  Note: You can use this only for demo purpose since the minimum requirement for Multi Node Kubernetes Cluster is :
  * Minimum 2 CPUs in each VM and
  * Minimum 2 GB RAM in each VM,

  which is not provided by this type (t2.micro). However, if you want better performance and you are willing to pay, you can choose other instance types according to your requirement. Since our instance type doesn't meet the minimum requirements, we have provided a work-around.
- Click on Next. And,
  * Choose the number of instances as 3
  * Select above created VPC under **Network** option.
  * Select above created Subnet under **Subnet** option.
  * Enable **Auto-assign Public IP**

  And, Click on Next.
- Select the SSD storage according to your requirement. Again, maximum of 30 GiB can be chosen per instance in free-tier. Click on Next and go to **Configure Security Group**
- Choose **Select an existing security group** under **Assign a security group** option and select above created security group.
- Download the Key-Pair.
- Once the instances are created, go to the instances and click on Connect (X2).

- Once the terminal is open after Connect, Install Docker and implement below steps on **all** instances

Install Docker

```
# Update System Packages
sudo apt-get update
# Install Required packages
sudo apt-get install apt-transport-https ca-certificates
    curl gnupg-agent software-properties-common
# Add Dockers official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
    sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
# SET UP THE REPOSITORY
sudo add-apt-repository "deb [arch=amd64] https://download.
    docker.com/linux/ubuntu $(lsb_release -cs) stable"
# Update System Packages
sudo apt-get update
# INSTALL DOCKER ENGINE
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- Install Kubernetes and implement below steps on **all** the instances [6].

Install Kubernetes

```
# Get the Kubernetes gpg key
curl -s https://packages.cloud.google.com/apt/doc/apt-key.
    gpg | sudo apt-key add -
# Add the Kubernetes repository
cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.
    list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
# Update System Packages
sudo apt-get update
# Install kubelet, kubeadm and kubectl
sudo apt-get install -y kubelet kubeadm kubectl
# Hold the versions of Docker, kubelet, kubeadm and kubectl
sudo apt-mark hold docker-ce kubelet kubeadm kubectl
# Add the iptables rule to sysctl.conf
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /
    etc/sysctl.conf
# Enable the iptables
sudo sysctl -p
```

- Execute below steps **only on Master Node**. As mentioned in the earlier steps, the work-around is provided in the first command with the flag **–ignore-preflight-errors** which ignores the configuration and runs anyway.

Start the Cluster

```
# Initialize the cluster
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU,Mem
# Set up local kubeconfig
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
# Apply Flannel CNI network overlay
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

- Execute the complete ”kubeadm join” command generated as output of above command **only on all Worker nodes.**
- Execute the commands from sections ’Common Steps for Kubernetes’, ’Common Steps for OpenFaaS’ and ’OpenFaaS’ sequentially **only on Master node.**
- Similar to Minikube process, you can access the OpenFaaS portal directly through the browser using the OPENFAAS_URL. In Custom made Kubernetes Cluster, the URL is as follows:
  PUBLIC_IP_OF_MASTER_NODE:GATEWAY_EXTERNAL_PORT_NO
- Moreover, as we are dealing with multi-node architecture, we can also view the services running on each node. Since the master node doesn’t possess any service but it splits the tasks among the worker nodes. This command depicts the same and shows what resources/containers are running on each node and demonstrates the utilization of multi-node architecture as shown in figure 15.

List services on nodes

```
kubectl get pods -n openfaas -o wide --field-selector spec.nodeName=NODE_NAME_HERE
```

```
ubuntu@ip-10-0-1-26:~$ kubectl get pods -n openfaas -o wide --field-selector spec.nodeName=ip-10-0-1-26
No resources found in openfaas namespace.
ubuntu@ip-10-0-1-26:~$ kubectl get pods -n openfaas -o wide --field-selector spec.nodeName=ip-10-0-1-154
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE             NOMINATED NODE   READINESS GATES
alertmanager-644957df4b-mv8w5       1/1     Running   0          52m   10.244.1.7   ip-10-0-1-154    <none>           <none>
basic-auth-plugin-bc899c574-6bjfw   1/1     Running   0          52m   10.244.1.2   ip-10-0-1-154    <none>           <none>
gateway-554f96996b-92f8h            2/2     Running   1          52m   10.244.1.5   ip-10-0-1-154    <none>           <none>
nats-7d86c64647-2kz5b               1/1     Running   0          52m   10.244.1.6   ip-10-0-1-154    <none>           <none>
queue-worker-77f4446d48-mdk4x       1/1     Running   1          52m   10.244.1.3   ip-10-0-1-154    <none>           <none>
queue-worker-77f4446d48-srxnw       1/1     Running   1          52m   10.244.1.4   ip-10-0-1-154    <none>           <none>
ubuntu@ip-10-0-1-26:~$ kubectl get pods -n openfaas -o wide --field-selector spec.nodeName=ip-10-0-1-227
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE             NOMINATED NODE   READINESS GATES
gateway-554f96996b-2w7sn      2/2     Running   1          52m   10.244.2.2   ip-10-0-1-227    <none>           <none>
prometheus-77f65d99b5-b2l6h   1/1     Running   0          52m   10.244.2.3   ip-10-0-1-227    <none>           <none>
ubuntu@ip-10-0-1-26:~$
```

Fig. 15. Workload on Worker Nodes

3.2.2 Using Managed Service - AWS EKS

We can also create a multi node kubernetes cluster using a managed service. We have opted for the flexible service provided by AWS called EKS (Elastic Kubernetes Service) [7]. EKS helps us to create and manage a cluster with automatic scaling.

- Before diving straight-away into EKS, lets first install the pre-requisites

Pre-Requisites

```
# Install eksctl CLI
curl --silent --location "https://github.com/weaveworks/
    eksctl/releases/latest/download/eksctl_$(uname -s)_amd64
    .tar.gz" | tar xz -C /tmp
# Move the downloaded package to bin folder
mv /tmp/eksctl /usr/local/bin
# Check Version
eksctl version
# Install and Configure AWS CLI - Refer Section 3.2.1
# Install Kubernetes CLI
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.
    k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
curl -LO https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/
    release/stable.txt)/bin/linux/amd64/kubectl.sha256
echo "$(<kubectl.sha256) kubectl" | sha256sum --check
# Install Helm CLI and OpenFaaS CLI - Refer Section 'Common
    Steps for OpenFaaS'
```

- Create the Cluster.
  Note: EKS automatically assigns **m5.large** type of instances to the cluster, which is a paid service. It can also be configured as per the requirement in the **create cluster** command. The steps mentioned below are depicted in figure 16 and 17 respectively.

Create EKS Cluster

```
eksctl create cluster --name=openfaas-eks-demo --nodes=2 --
    auto-kubeconfig --region=eu-central-1
# Set KUBECONFIG
export KUBECONFIG=~/.kube/eksctl/clusters/openfaas-eks-demo
# Check the nodes
kubectl get nodes
```

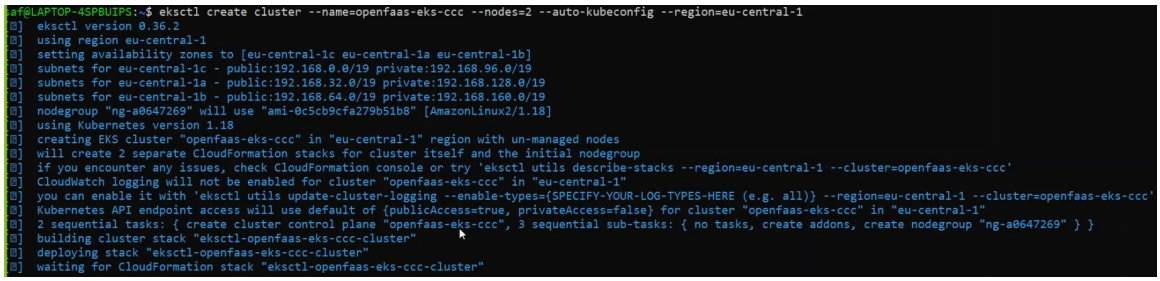

Fig. 16. Create the Cluster

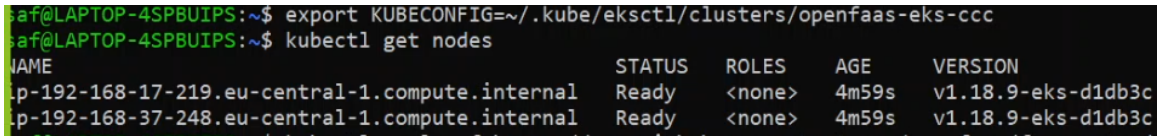

Fig. 17. List the Nodes

- Now, execute all the commands from the section 'Common Steps for OpenFaaS'. And, continue execution from next section.

3.2.3 OpenFaaS

- Install and Create OpenFaaS deployments [7]. The output for below command should be similar to figure 18.

Install OpenFaaS

```
helm upgrade OpenFaaS
--install openfaas/openfaas \
--namespace openfaas \
--set functionNamespace=openfaas-fn \
--set serviceType=LoadBalancer \
--set basic_auth=true \
--set operator.create=true \
--set gateway.replicas=2 \
--set queueWorker.replicas=2
```

```
af@LAPTOP-4SPBUIPS:~$ helm repo add openfaas https://openfaas.github.io/faas-netes/
openfaas" already exists with the same configuration, skipping
af@LAPTOP-4SPBUIPS:~$ helm upgrade openfaas --install openfaas/openfaas \
> --namespace openfaas \
> --set functionNamespace=openfaas-fn \
> --set serviceType=LoadBalancer \
> --set basic_auth=true \
> --set operator.create=true \
> --set gateway.replicas=2 \
> --set queueWorker.replicas=2
elease "openfaas" does not exist. Installing it now.
AME: openfaas
AST DEPLOYED: Mon Jan 25 00:40:58 2021
AMESPACE: openfaas
TATUS: deployed
EVISION: 1
EST SUITE: None
OTES:
o verify that openfaas has started, run:

  kubectl -n openfaas get deployments -l "release=openfaas, app=openfaas"
```

Fig. 18. OpenFaaS Deployments

- Here, URL for OpenFaas can be fetched by executing **kubectl get service** command for gateway external, then the fetched information is stored in the variable. This step is illustrated in figure 19.

OpenFaaS Login

```
# Set OpenFaaS URL
export OPENFAAS_URL=$(kubectl get svc -n openfaas gateway-
    external -o jsonpath='{.status.loadBalancer.ingress[*].
    hostname}'):8080 \
&& echo Your gateway URL is: $OPENFAAS_URL
# Login
echo $PASSWORD | faas-cli login --username admin --password
    -stdin
```

```
af@LAPTOP-4SPBUIPS:~$ export OPENFAAS_URL=$(kubectl get svc -n openfaas gateway-external -o jsonpath='{.status.loadBalancer.ingress[*].hostname}'):8080 \
> && echo Your gateway URL is: $OPENFAAS_URL
our gateway URL is: ab52c5603ad0e4834b19d2cf70249e77-1961071324.eu-central-1.elb.amazonaws.com:8080
af@LAPTOP-4SPBUIPS:~$ kubectl get deployments -n openfaas
AME              READY   UP-TO-DATE   AVAILABLE   AGE
lertmanager       1/1     1            1           70s
asic-auth-plugin  1/1     1            1           70s
ateway            2/2     2            2           70s
ats               1/1     1            1           70s
rometheus         1/1     1            1           70s
ueue-worker       2/2     2            2           70s
af@LAPTOP-4SPBUIPS:~$ echo $PASSWORD | faas-cli login --username admin --password-stdin
alling the OpenFaaS server to validate the credentials...
ARNING! Communication is not secure, please consider using HTTPS. Letsencrypt.org offers free SSL/TLS certificates.
redentials saved for admin http://ab52c5603ad0e4834b19d2cf70249e77-1961071324.eu-central-1.elb.amazonaws.com:8080
af@LAPTOP-4SPBUIPS:~$ kubectl create deployment faasml --image="docker.io/sdgamer007/faasml:latest" -n openfaas
eployment.apps/faasml created
af@LAPTOP-4SPBUIPS:~$ kubectl expose deployment/faasml --type=LoadBalancer --port=5000 -n openfaas
ervice/faasml exposed
af@LAPTOP-4SPBUIPS:~$ kubectl get service faasml -n openfaas
AME     TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)          AGE
aasml   LoadBalancer   10.100.196.47   acade53eb15934e67b9323689157fd4d-1144214071.eu-central-1.elb.amazonaws.com   5000:31668/TCP   12s
```

Fig. 19. Login to FaaS-CLI

4. Operation

4.1 Lifecycle of Functions

Every serverless function goes through certain phases of its lifecycle. This section illustrates how to create, build, push, deploy, manage and use functions by listing the most common and important commands with examples. The functions can be created in **csharp, go, java11, node, php7, python3 and ruby** programming languages.

- **Create:** This command will create a directory named 'hello' in your current directory along with hello.yml and template directory. hello.yml contains language python3, and image information. Image name can be preceded with docker id where you want to store the image. The project is available in the hello directory. Handler.py and requirements.txt are the files for python, but if you consider java, then the package will have the source project and Handler.java file that contains the main class. So we can generate a small project and deploy it as a function. This step is demonstrated in figure 20.

Create

```
faas-cli new --lang python3 hello
```

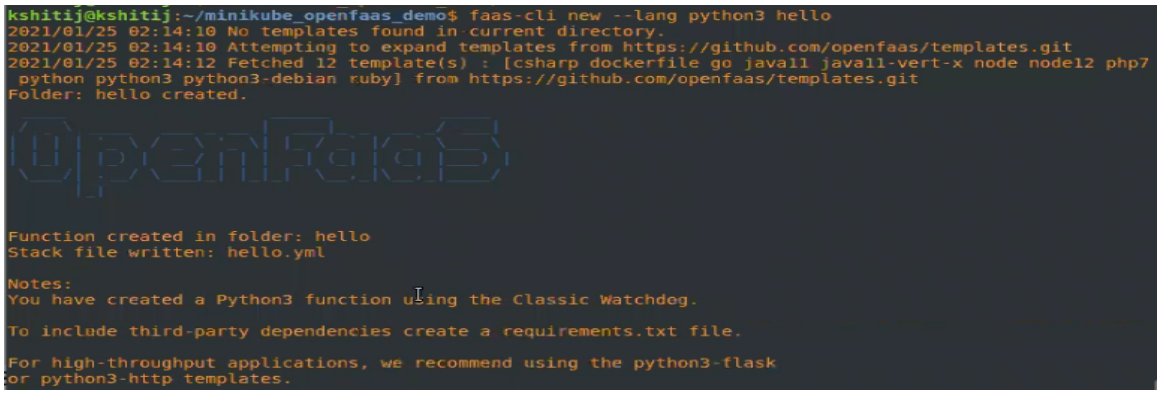

Fig. 20. Create New Project

- **Build:** This command builds the project, creates the package for deployment and creates the docker image as depicted in figure 21.

Build

```
faas-cli build -f hello.yml
```

```
kshitij@kshitij:~/minikube_openfaas_demo$ faas-cli build -f hello.yml
[0] > Building hello.
Clearing temporary build folder: ./build/hello/
Preparing: ./hello/ build/hello/function
Building: kshitijgit123/hello:latest with python3 template. Please wait..
Sending build context to Docker daemon  9.216kB
Step 1/31 : FROM --platform=${TARGETPLATFORM:-linux/amd64} ghcr.io/openfaas/classic-watchdog:0.1.4 as wa
tchdog
0.1.4: Pulling from openfaas/classic-watchdog
5c03a2f5c550: Pulling fs layer
5c03a2f5c550: Verifying Checksum
5c03a2f5c550: Download complete
5c03a2f5c550: Pull complete
```

Fig. 21. Build the Project

Make sure at this point, that you have logged in to your docker account. The following command authenticates your docker credentials:

Docker Login

```
docker login
```

- **Push:** This command pushes the created image to docker registry and creates a new repository of the image on docker.

  Push

  ```
  faas-cli push -f hello.yml
  ```

- **Deploy** This command deploys the function to OpenFaaS. Here, we need to provide the gateway URL as shown in figure 22.

  Depoy

  ```
  faas-cli deploy -f hello.yml --gateway http://$(minikube ip
      ):31112
  ```

```
kshitij@kshitij:~/minikube_openfaas_demo$ faas-cli deploy -f hello.yml --gateway http://$(minikube ip):3
1112
Deploying: hello.
WARNING! Communication is not secure, please consider using HTTPS. Letsencrypt.org offers free SSL/TLS c
ertificates.

Deployed. 202 Accepted.
URL: http://192.168.49.2:31112/function/hello.openfaas-fn
```

Fig. 22. Deploy the Project

Note: Instead of running above 3 commands, we can also use below command to build, push and deploy a function in a single run:
faas-cli up -f hello.yml –gateway http://$(minikube ip):31112

- **Invoke:** Now, we are ready to use the deployed function. We can invoke the function through CLI using **cURL** and **faas-cli** as given below. One thing to note here is, the gateway URL belongs to Minikube, but it will be different in other cases as described similarly in above sections for OPENFAAS_URL.

Invoke

```
# faas-cli
echo "Hello World" | faas-cli invoke hello --gateway http
    ://$(minikube ip):31112

# cURL
curl -X POST --data "Hello World"  http://$(minikube ip)
    :31112/function/hello
```

4.2 OpenFaaS Store

OpenFaaS also provides a functional store where various pre-built functions are available and can be easily deployed and used in an application.

OpenFaaS Store commands

```
# List available functions
faas-cli store list
# Show information about a function of the store
faas-cli store inspect <function_name>
# Deploy a function from the store
faas-cli store deploy <function_name>
```

Although these functions can be introduced from the CLI, but it is much easier with the OpenFaaS UI, which is also included in this documentation.

4.3 A Simple Web Application - ML FaaS

We have built a simple web application to demonstrate the use of OpenFaaS functions which comprises several functions from the OpenFaaS store and a custom made function as well. The two main options for creating your function are through the CLI or the UI. This documentation will cover both options. Our application provides a way to use the OpenFaaS pre-built machine learning capabilities. The project aims to use a serverless approach and execute all those machine learning algorithms in a serverless way rather than

using the traditional server model. We can deploy OpenFaaS on any machine which has Docker and container orchestration tool, or we can use a cloud service. For example, The following problems are solved with this approach.

- Version control.
- Even if every application is package dependent, but every application has its own docker file that helps to create unique platform for the function.
- Scalable - Kubernetes helps to achieve scalability.
- Every model being independent of each other, failure of one model won't affect other models.

The application can be deployed by the following command. Initially, we need to create a deployment in Kubernetes, preferred in the same namespace. Since we have already changed the default namespace to 'openfaas', so there is no need to mention the namespace flag in the command now. After the creation of deployment, expose the deployment to port 5000 so that it will be accessible to use.

Create deployment and expose the application 'faasml'

```
kubectl create deployment faasml --image="docker.io/sdgamer007/faasml:latest"
kubectl expose deployment/faasml --type=LoadBalancer --port=5000
```

We have included following ML functions in our application:

- **Face Blur** We have deployed the image directly from the public docker repository. While deploying we need to pass image and name arguments. While illustrating the lifecycle above, We have deployed the function **hello** through yml file.

  Pull and Deploy Faceblur image

  ```
  docker pull esimov/pigo-openfaas-faceblur:0.1
  faas-cli deploy --image=esimov/pigo-openfaas-faceblur --name faceblur
  ```

- **Inception** It is also possible to deploy function directly from the store through CLI.

  Deploy Inception function

  ```
  faas-cli store deploy inception
  ```

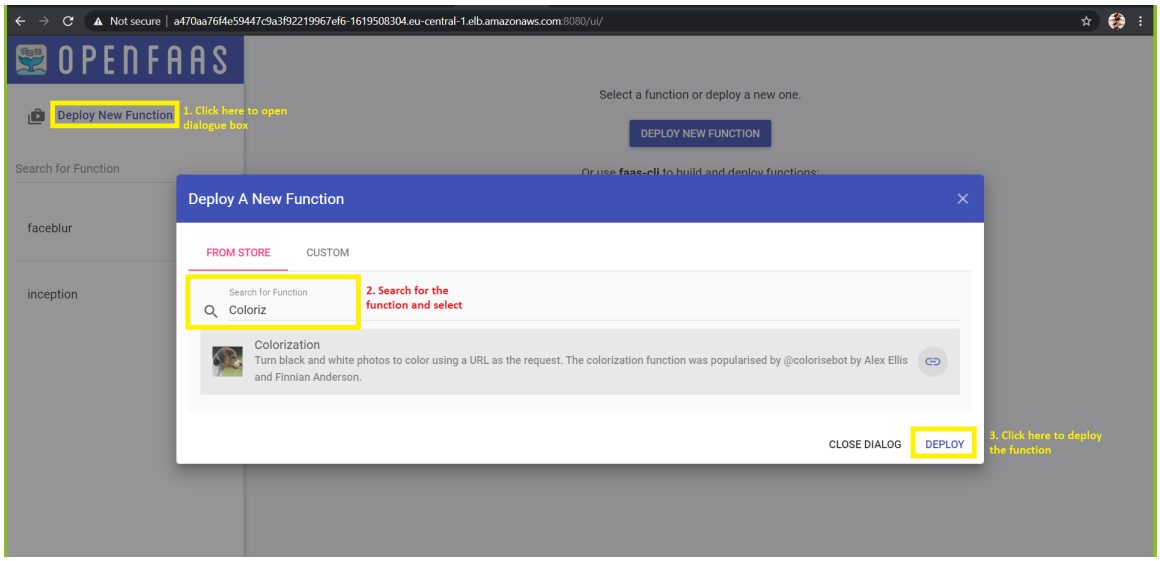

Fig. 23. Deploying a function via OpenFaas UI

- **Colorization:** Let's deploy a machine-learning function called **Colorization** through OpenFaaS UI as shown in the figure 23.
- **Coherent Line Drawing:** Like Colorization function deployment, search for "line drawing" and deploy the function.
- **Face Detection by Pigo:** Like Colorization function deployment, search for "face detect" and deploy the function.

Hereby, we have described all the ways of deploying a function, i.e. thorugh store (CLI and UI), directly from the docker repository and yml file.

Application Snapshots: Once the functions are deployed, they are ready to use. You will find the image links in the references so you can try the same link. You can also use other images, just try with lower resolution images otherwise the output won't be generated. The images in figure 24, 25, 26, 27 and 28 are the captures of our application.

- **Face Blur [8]:**

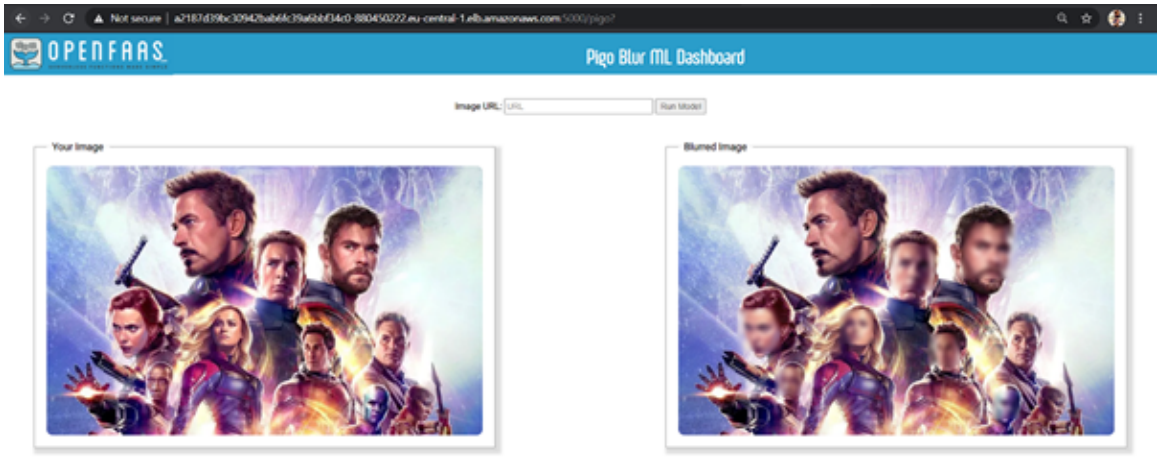

Fig. 24. Face Blur

- **Inception [9]:**

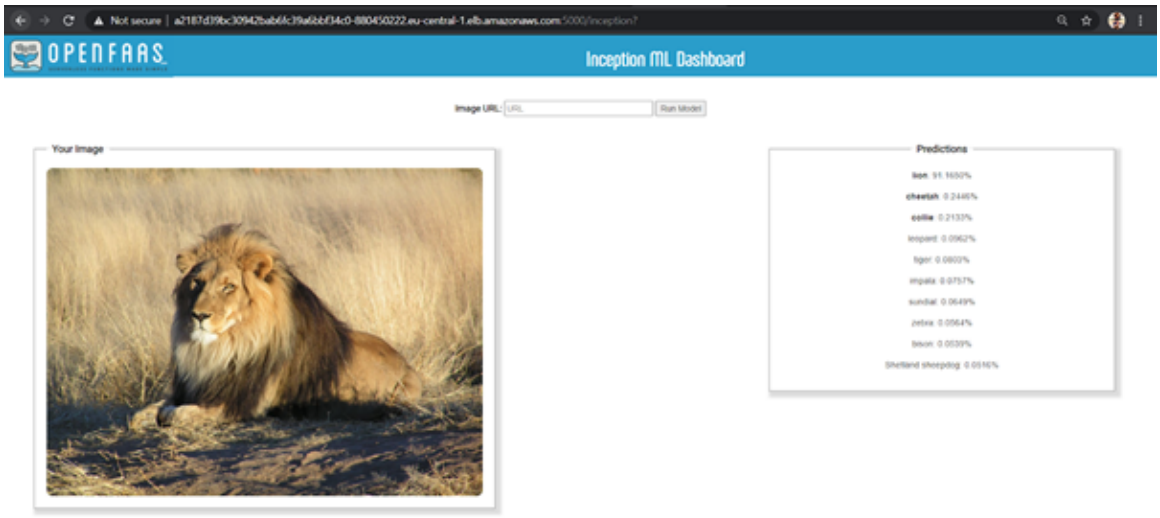

Fig. 25. Inception

- **Colorization [10]:**

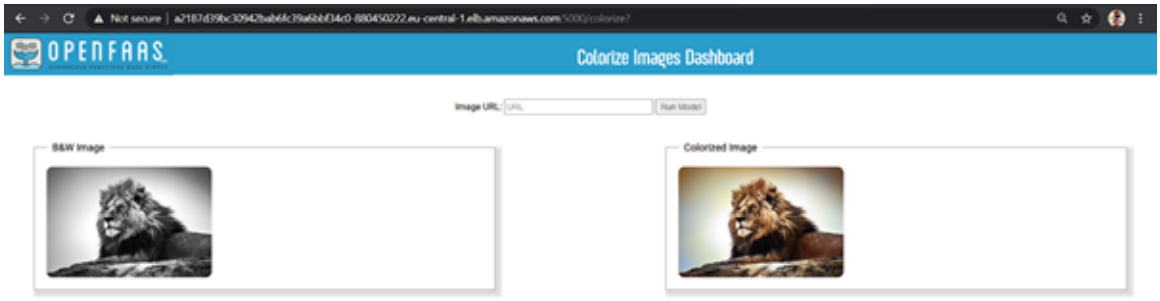

Fig. 26. Colorization

- **Coherent Line Drawing [11]:**

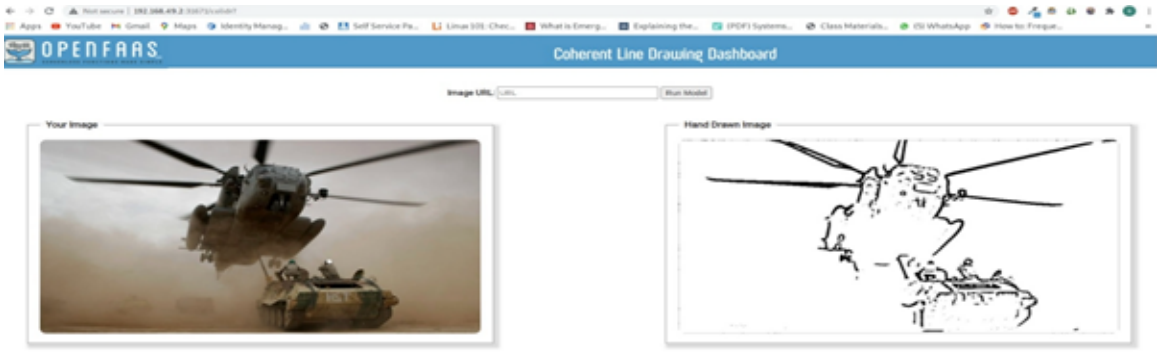

Fig. 27. Coherent Line Drawing

- **Face Detect [12]:**

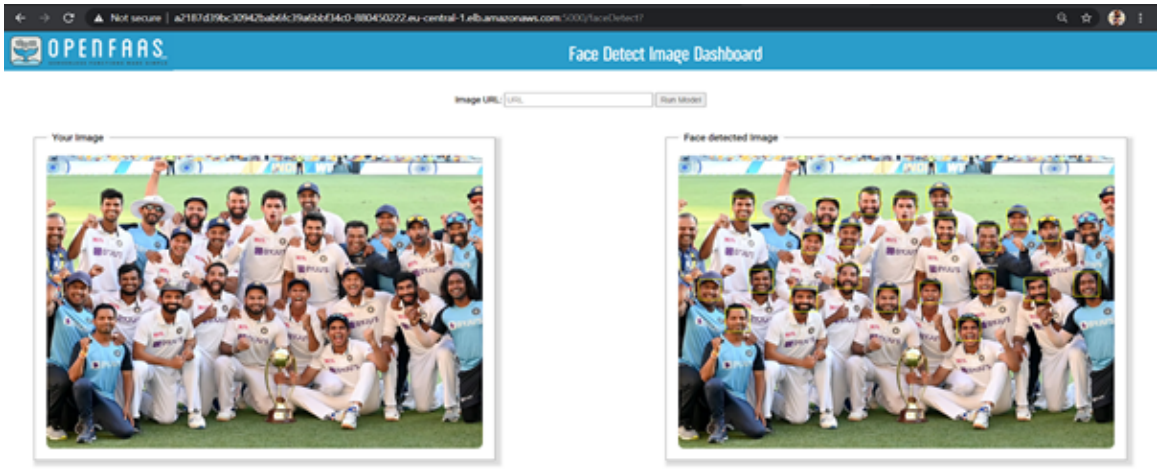

Fig. 28. Face Detect

4.4 Monitoring of Functions

The OpenFaaS Gateway collects metrics on several existing replicas of the functions, how often they are invoked, their HTTP codes (success/failure), and the latency of each request. We can view this data in the OpenFaaS UI, or via the command **faas-cli list**, but the most effective way to monitor the data is through a dashboard using Grafana.

- **Prometheus:** It is an open-source event monitoring and alerting tool based on time-series database. It collects metrics which are available via the Gateway's API for auto-scaling. It allows us to view the details in both console and graph as shown in figure 29 and 30.

Prometheus Setup

```
# Expose Deployment
kubectl expose deployment prometheus --type=NodePort --name
    =prometheus-ui
# View the prometheus-ui service
kubectl get svc prometheus-ui
# Open prometheus-ui on local system
kubectl port-forward svc/prometheus-ui 9090:9090 &
```

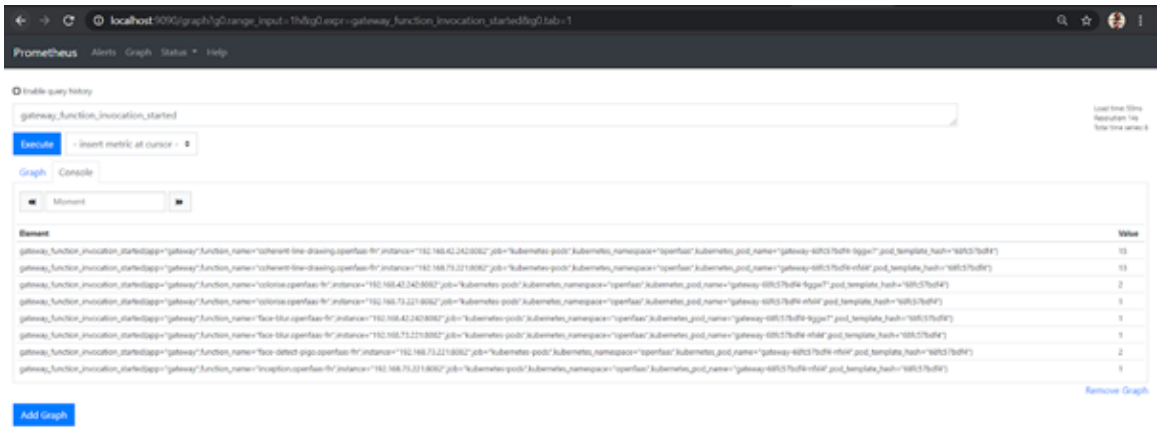

Fig. 29. Prometheus Console

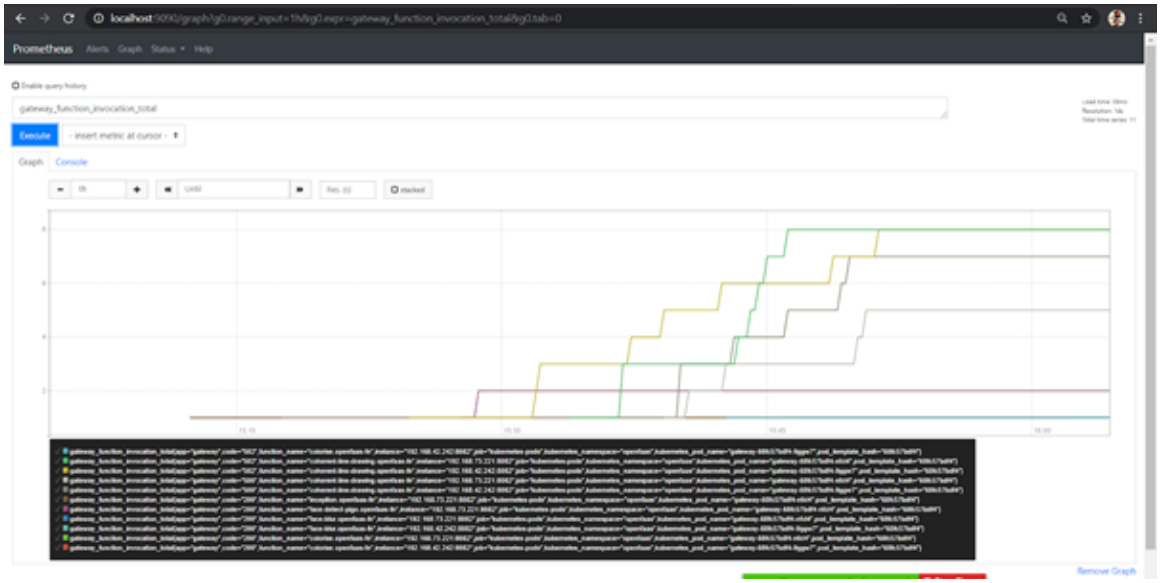

Fig. 30. Prometheus Graph

- **Grafana:** It is an open-source visualization software, which helps the users to understand the complex data with the help of data metrics fetched by Prometheus. This tool can be used for data analytics, to make cool dashboards and highly customizable. The data can be visualized for a particular period and can be tuned to fetch metrics with tunable refresh time rates. The dashboard of Grafana is depicted in figure 31.

Grafana Setup

```
# Create Grafana pod
kubectl run grafana --image=stefanprodan/faas-grafana:4.6.3
    --port=3000
# Expose the pod
kubectl expose pod grafana --type=NodePort --name=grafana
# View the Grafana service
kubectl get service Grafana
# Open Grafana on local system
kubectl port-forward svc/grafana 3000:3000 &
```

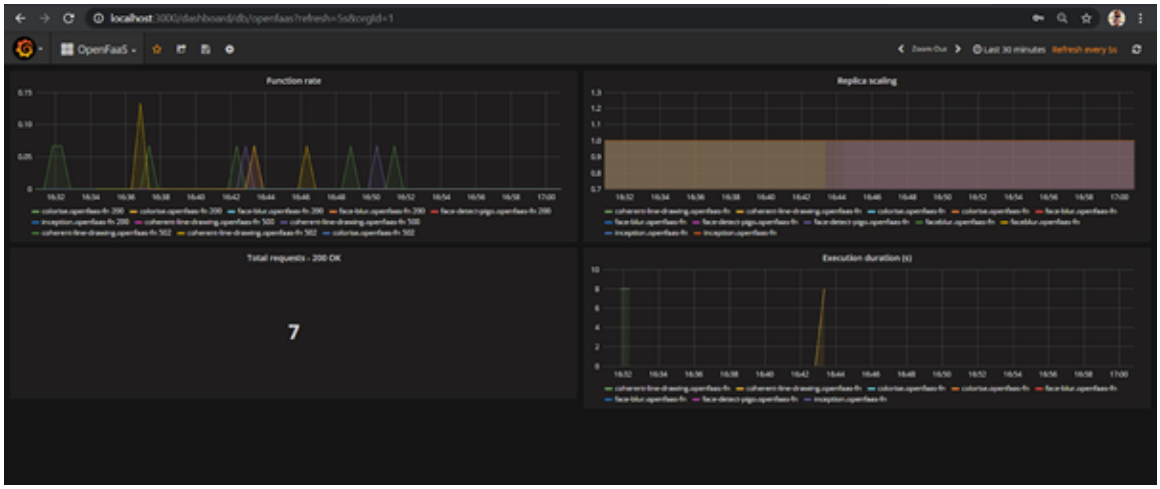

Fig. 31. Grafana Dashboard

5. Conclusion

In this project, we have implemented OpenFaaS using several ways. Moreover, we have successfully built an application to demonstrate the use of OpenFaaS on single-node and multi-node Kubernetes cluster since OpenFaaS can be integrated with any cloud-orchestration tool. With Kubernetes, we can build productionized solutions. However, some points need to be considered while working with OpenFaaS, such as the communication with the gateway is not encrypted, both via UI and command line. But, we can use the ingress component of Kubernetes to make this secure as it uses SSL. Moreover, other things like disaster recovery, backups, and fault tolerance need to experiment before using in production.

Acknowledgments

This research was possible under the guidance and direction of Prof. Dr. Christian Baun. This report is the final submission for the course Cloud Computing (CC) taught in the Master's programme High Integrity Systems (HIS).

References

[1] Gaunt S What you need to know about building serverless architectures. Available at https://maxkelsen.com/blog/building-serverless-architectures.

[2] OpenFaaS Cloud Architecture. Available at https://docs.openfaas.com/openfaas-cloud/architecture/.

[3] OpenFaaS Architecture Gateway. Available at https://docs.openfaas.com/architecture/gateway/.

[4] Mutai J (2020) How to install minikube on ubuntu 20.04/18.04 debian 10 linux. Available at https://computingforgeeks.com/how-to-install-minikube-on-ubuntu-debian-linux/.

[5] Ellis A (2017) Getting started with openfaas on minikube. Available at https://medium.com/faun/getting-started-with-openfaas-on-minikube-634502c7acdf.

[6] Shivalkar R Setup a kubernetes cluster on aws ec2 instance with ubuntu using kubeadm. Available at https://www.howtoforge.com/setup-a-kubernetes-cluster-on-aws-ec2-instance-ubuntu-using-kubeadm/.

[7] Hein C (2018) Aws eks. Available at https://aws.amazon.com/blogs/opensource/deploy-openfaas-aws-eks/.

[8] Face Blur image - Avengers. Available at https://cnet4.cbsistatic.com/img/j7SdHs9Ac8coHkwTOcJG1DYcQI4=/940x0/2019/04/19/f20d0d6a-1781-49a4-90ab-e285109b65b2/avengers-endgame-imax-poster-crop.png.

[9] Inception Image - Lion. Available at https://upload.wikimedia.org/wikipedia/commons/7/73/Lion_waiting_in_Namibia.jpg.

[10] Colorization Image - Lion. Available at https://blinq.art/blog/wp-content/uploads/2018/04/blinq-art-black-white-default.jpg.

[11] Line Drawing Image - Helicopter. Available at https://hdwallpaperim.com/wp-content/uploads/2017/08/24/103253-helicopters-MH-53_Pave_Low-748x421.jpg.

[12] Face Detection Image - Team India. Available at https://gumlet.assettype.com/bloombergquint%2F2021-01%2F4ae730ce-17e5-4438-937e-17c4d27900f2%2FWhatsApp_Image_2021_01_19_at_14_53_47.jpeg?rect=32%2C0%2C1196%2C861&auto=format%2Ccompress&w=1200.