CI/CD of Cloud Functions including the Service by using Infrastructure as Code

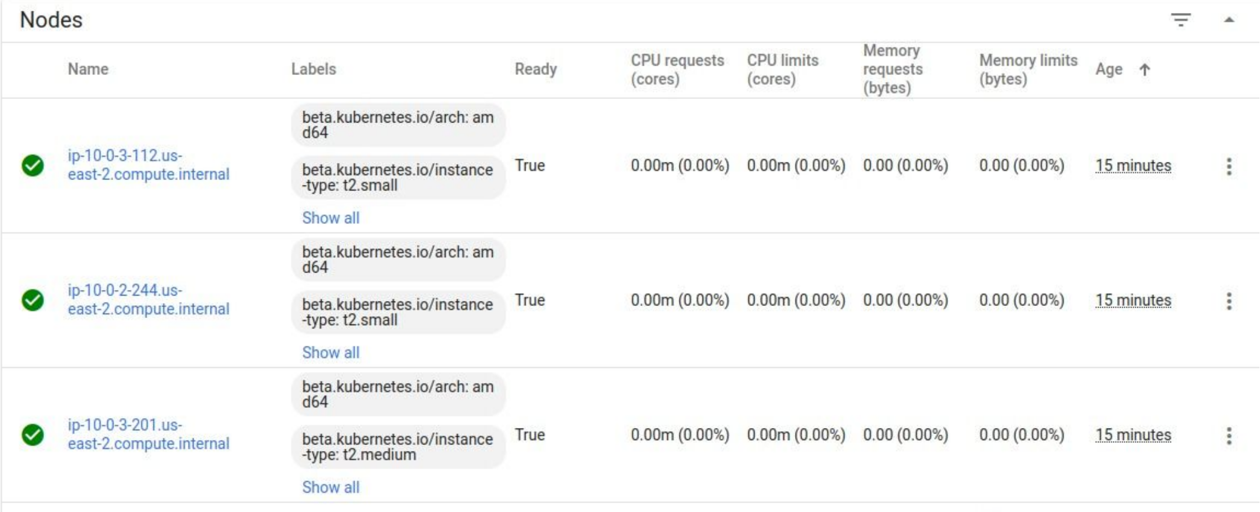
Cloud Computing SS2022

Table of Content

1. Introduction
2. Architecture
3. Installation
4. Pipelines

1. Introduction — Goal

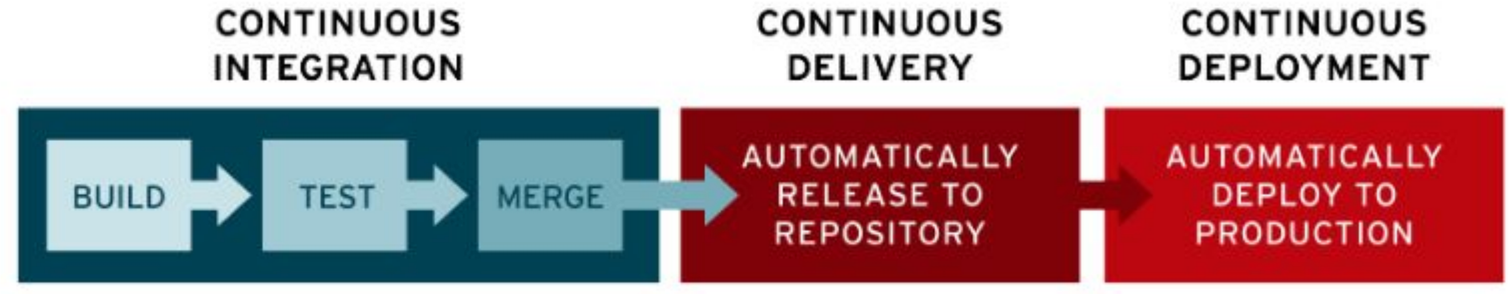
- Automated cluster creation and deployment
- Processes can be repeated and is reliable
- Deployment of OpenFaas Functions



Nodes

	Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age ↑
✓	ip-10-0-3-112.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.small Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	15 minutes
✓	ip-10-0-2-244.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.small Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	15 minutes
✓	ip-10-0-3-201.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.medium Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	15 minutes

1. Introduction — CI/CD



- Automates manual processes
- Reduces error proneness
- Reduces the likelihood of buggy versions going into production

https://www.redhat.com/en/topics/devops/what-is-ci-cd

1. Introduction — Terraform

- Infrastructur as Code Tool
- High-Level Configuration Language HCL (HashiCorp Configuration Language)

1. Developer describes the desired end-state for Cloud of on-premises infrastructure
2. Terraform generates a plan to reaching the end-state
3. Terraform executes the plan

https://www.ibm.com/cloud/learn/terraform#:~:text=Terraform%20is%20an%20open%20source%20%E2%80%9CInfrastructure%20as%20Code%E2%80%9D,cloud%20or%20on-premises%20infrastructure%20for%20running%20an%20application

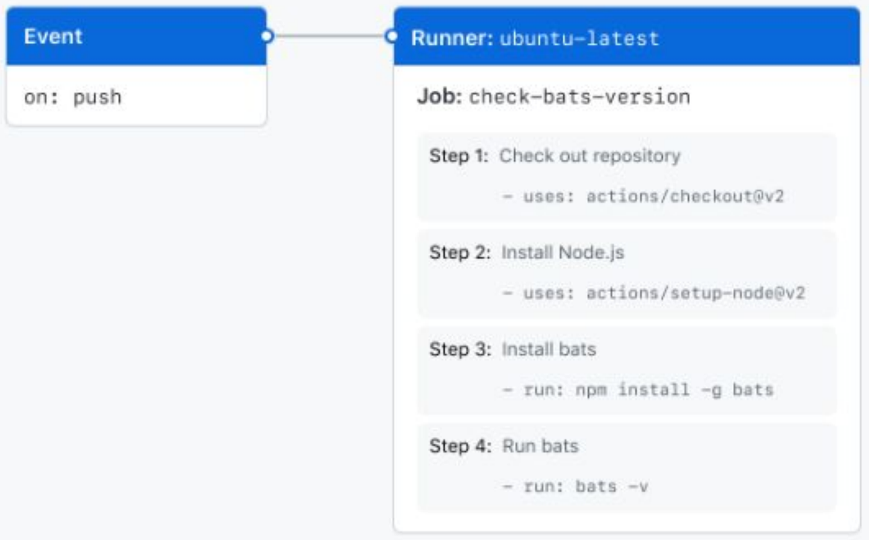
1. Introduction — Kubernetes

- Framework to manage containerized workloads and services
- Kubernetes offers the following features
  - Service discovery (expose Services through DNS)
  - Load balancing
  - Storage orchestration (automatically mount a storage system of choice)
  - Automated rollouts and rollbacks (automatically creates, deploy, removes containers)
  - Automatic bin packing (tell a kubernetes how much cpu and ram a container is allowed to use)
  - Self-healing (kills and restarts failed and unresponsive containers)

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

1. Introduction — Github Actions

- Github: the most popular version-control and collaboration platform
- Github Actions: Reusable event based workflows, which are divided into jobs, are executed on predefined events.
- Example Events:
    - push to repository, pull events, an issue being created or other workflow



Event

on: push

Runner: ubuntu-latest

Job: check-bats-version

Step 1: Check out repository

- uses: actions/checkout@v2

Step 2: Install Node.js

- uses: actions/setup-node@v2

Step 3: Install bats

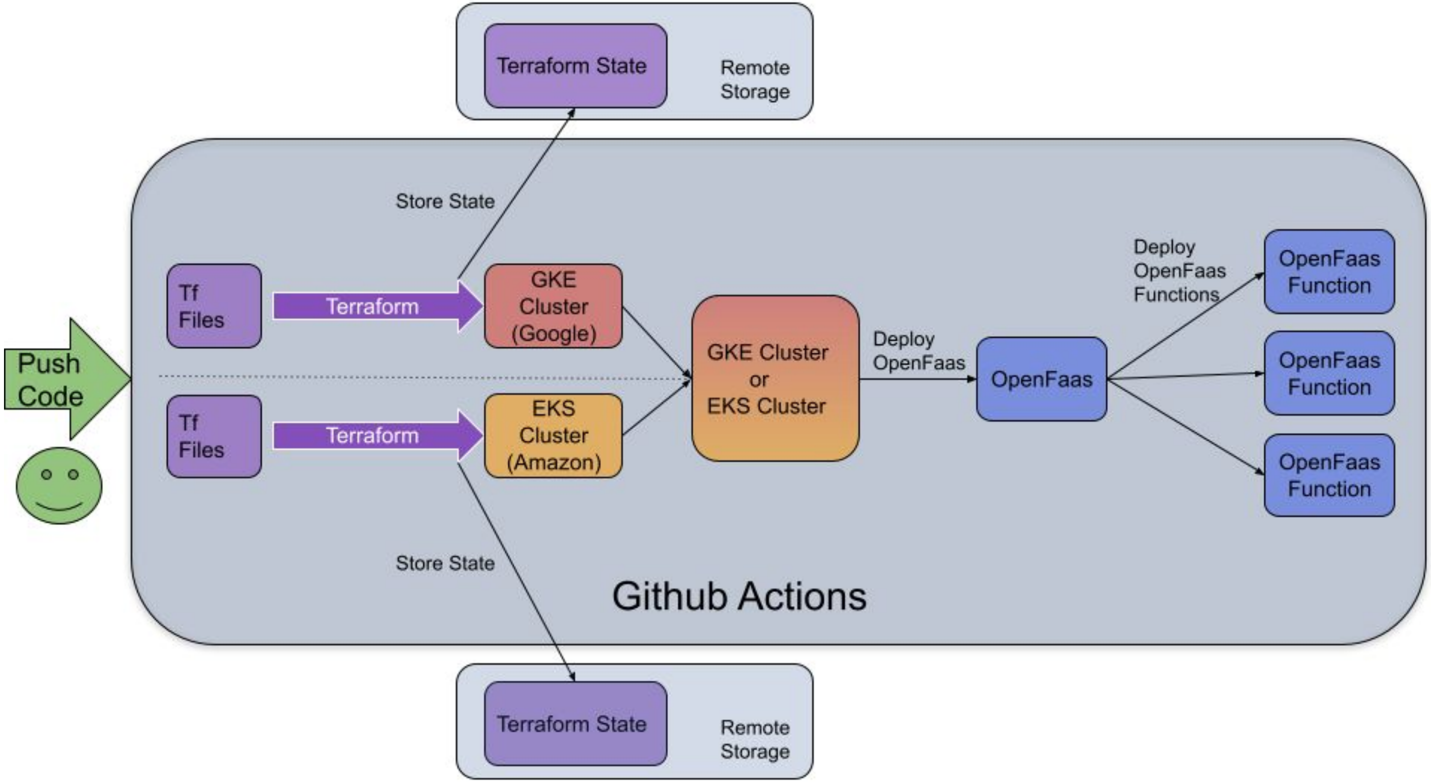
- run: npm install -g bats

Step 4: Run bats

- run: bats -v

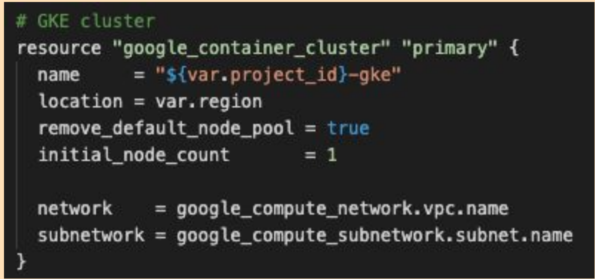
https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

2. Architecture

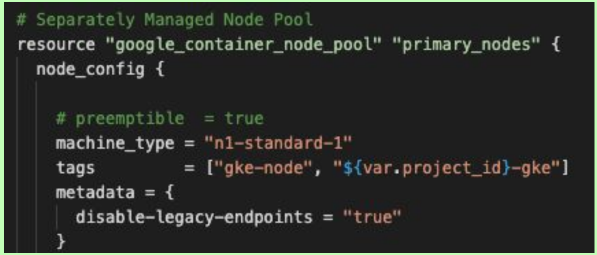


3. Deployment of GKE Cluster with Terraform

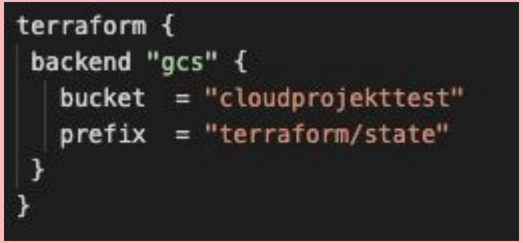
- Google Kubernetes Engine (GKE)
- Resource “google_container_cluster”
- Resource “google_container_node_pool”
- Store Terraform state in a remote storage



```
# GKE cluster
resource "google_container_cluster" "primary" {
  name     = "${var.project_id}-gke"
  location = var.region
  remove_default_node_pool = true
  initial_node_count       = 1

  network    = google_compute_network.vpc.name
  subnetwork = google_compute_subnetwork.subnet.name
}
```



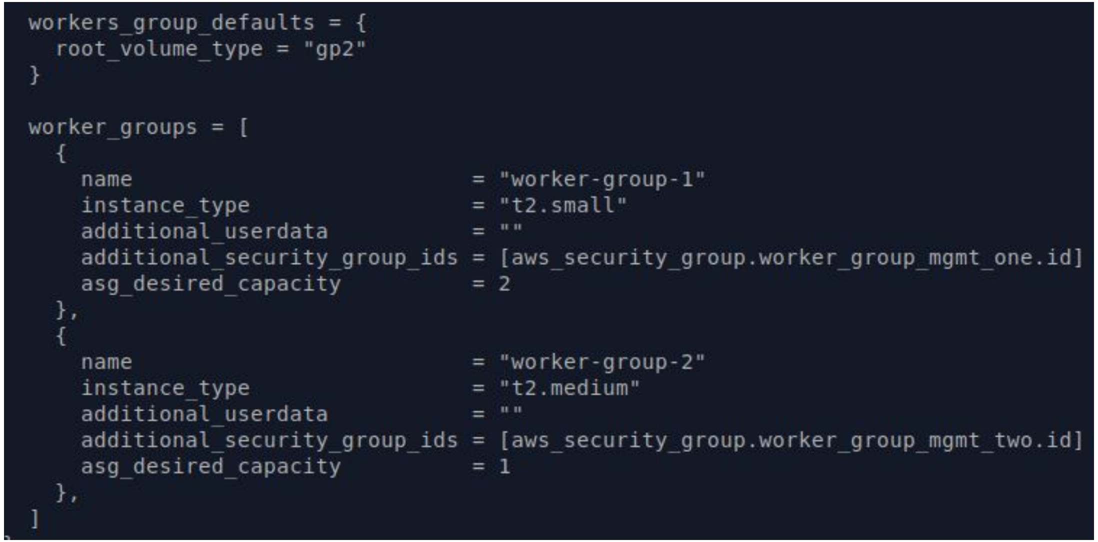
```
# Separately Managed Node Pool
resource "google_container_node_pool" "primary_nodes" {
  node_config {

    # preemptible  = true
    machine_type = "n1-standard-1"
    tags         = ["gke-node", "${var.project_id}-gke"]
    metadata = {
      disable-legacy-endpoints = "true"
    }
```



```
terraform {
  backend "gcs" {
    bucket  = "cloudprojekttest"
    prefix  = "terraform/state"
  }
}
```

3. AWS

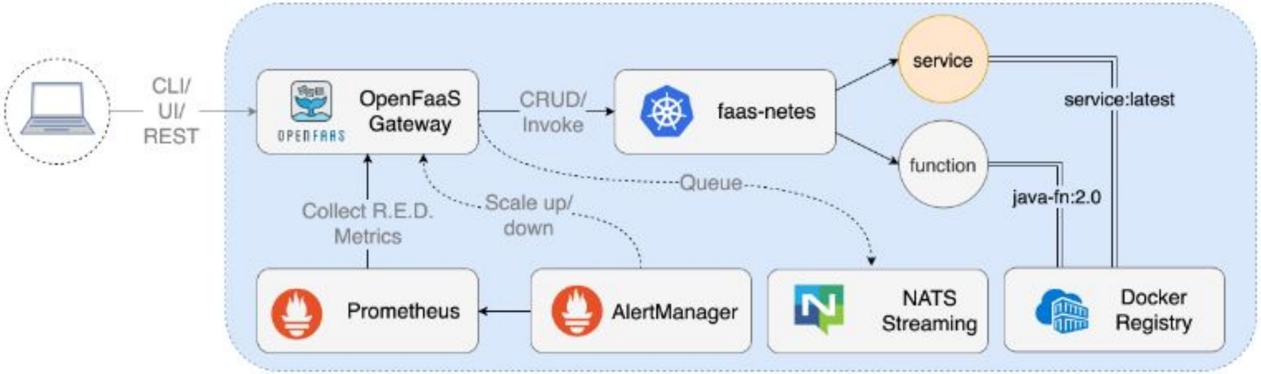
- root_volume_type: type of Storage pool gp2 = ssd’s only, gp3 = newer generation of ssd’s
- Instances describe how many CPU, Memory, Storage and Network speed is usable (1 vCPU, 2GiB RAM)
- desired_capacity: the initial capacity of the Auto Scaling group

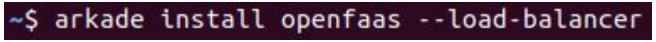


```
workers_group_defaults = {
  root_volume_type = "gp2"
}

worker_groups = [
  {
    name                          = "worker-group-1"
    instance_type                 = "t2.small"
    additional_userdata           = ""
    additional_security_group_ids = [aws_security_group.worker_group_mgmt_one.id]
    asg_desired_capacity          = 2
  },
  {
    name                          = "worker-group-2"
    instance_type                 = "t2.medium"
    additional_userdata           = ""
    additional_security_group_ids = [aws_security_group.worker_group_mgmt_two.id]
    asg_desired_capacity          = 1
  },
]
```

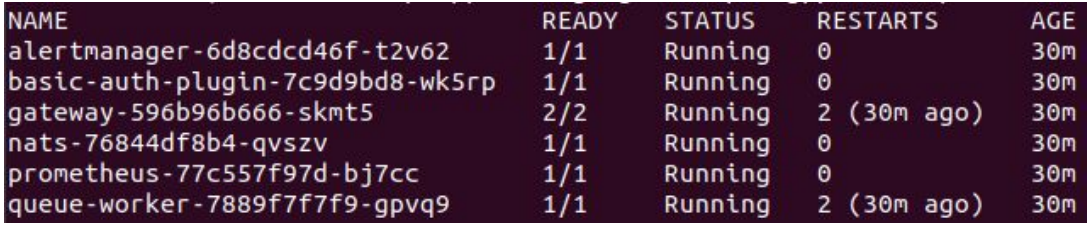
eks-cluster.tf

3. Installation Openfaas





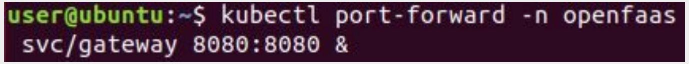
```
~$ arkade install openfaas --load-balancer
```



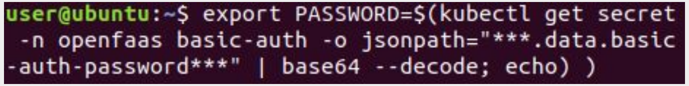
```
NAME                                READY   STATUS    RESTARTS      AGE
alertmanager-6d8cdcd46f-t2v62       1/1     Running   0             30m
basic-auth-plugin-7c9d9bd8-wk5rp    1/1     Running   0             30m
gateway-596b96b666-skmt5            2/2     Running   2 (30m ago)   30m
nats-76844df8b4-qvszv               1/1     Running   0             30m
prometheus-77c557f97d-bj7cc         1/1     Running   0             30m
queue-worker-7889f7f7f9-gpvq9       1/1     Running   2 (30m ago)   30m
```

3. Possible connections to Openfaas

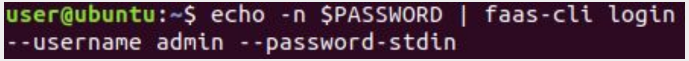
Port-forwarding

1. 

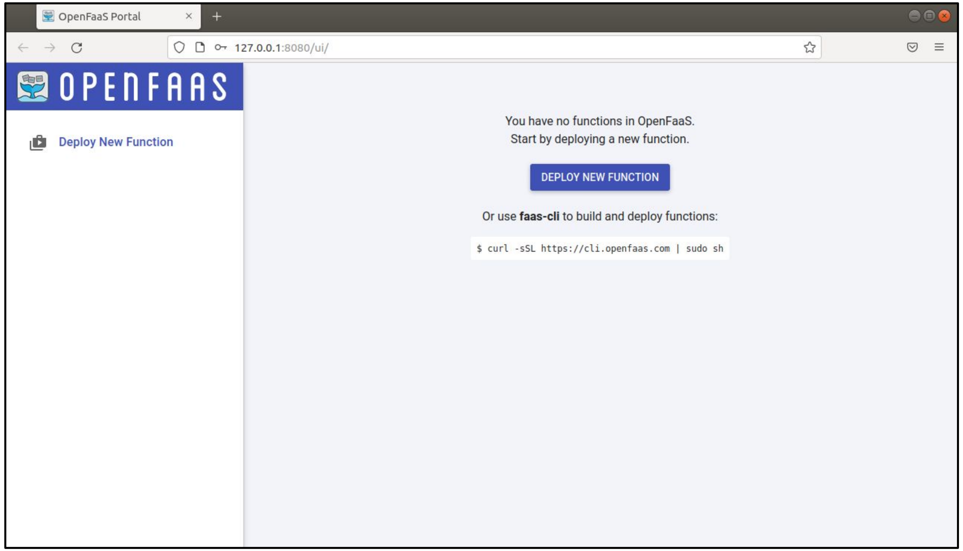
```
user@ubuntu:~$ kubectl port-forward -n openfaas
 svc/gateway 8080:8080 &
```

2. 

```
user@ubuntu:~$ export PASSWORD=$(kubectl get secret
 -n openfaas basic-auth -o jsonpath="***.data.basic
-auth-password***" | base64 --decode; echo) )
```

3. 

```
user@ubuntu:~$ echo -n $PASSWORD | faas-cli login
--username admin --password-stdin
```



You have no functions in OpenFaaS.
Start by deploying a new function.

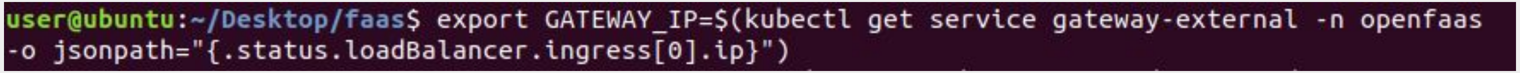
Or use **faas-cli** to build and deploy functions:

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

3. Possible connections to Openfaas

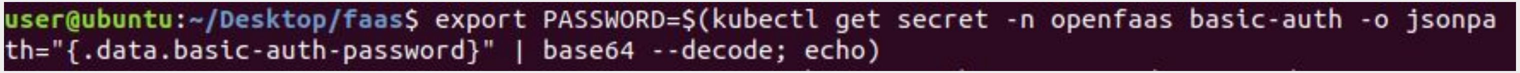
Connection over public IP:

1.



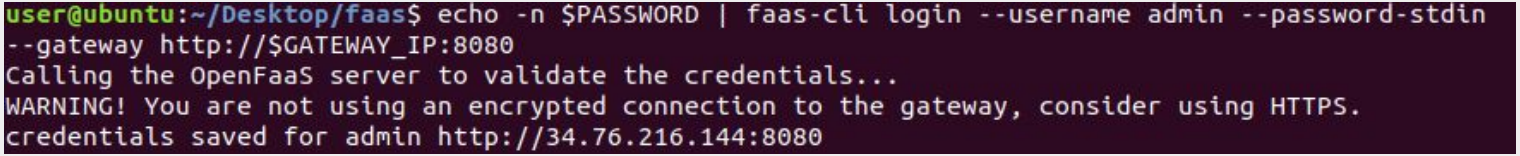
```
user@ubuntu:~/Desktop/faas$ export GATEWAY_IP=$(kubectl get service gateway-external -n openfaas
-o jsonpath="{.status.loadBalancer.ingress[0].ip}")
```

1.



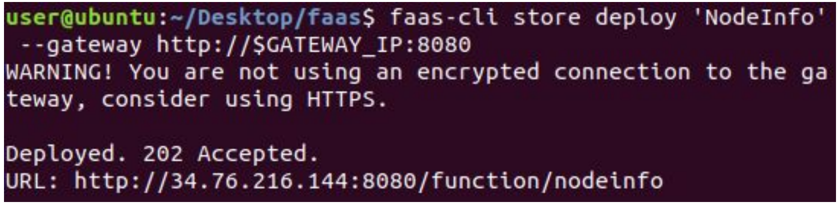
```
user@ubuntu:~/Desktop/faas$ export PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpa
th="{.data.basic-auth-password}" | base64 --decode; echo)
```

1.

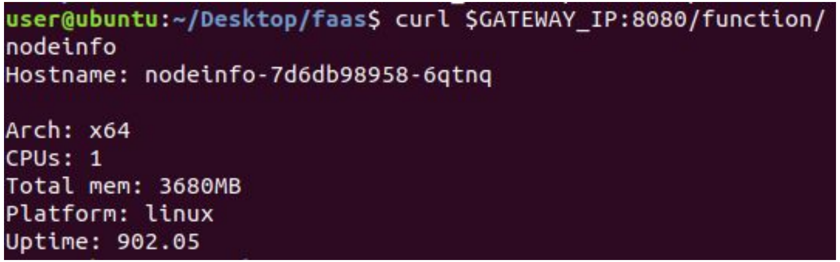


```
user@ubuntu:~/Desktop/faas$ echo -n $PASSWORD | faas-cli login --username admin --password-stdin
--gateway http://$GATEWAY_IP:8080
Calling the OpenFaaS server to validate the credentials...
WARNING! You are not using an encrypted connection to the gateway, consider using HTTPS.
credentials saved for admin http://34.76.216.144:8080
```

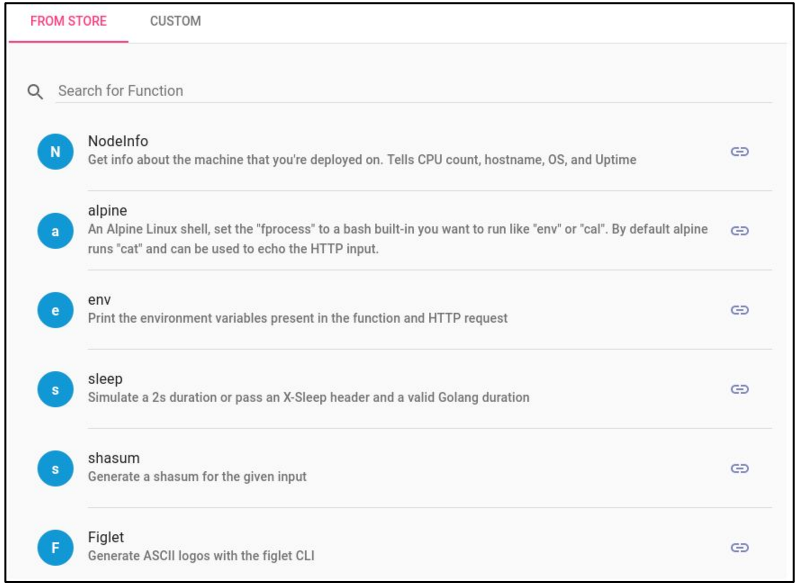
3. Function from store



```
user@ubuntu:~/Desktop/faas$ faas-cli store deploy 'NodeInfo'
 --gateway http://$GATEWAY_IP:8080
WARNING! You are not using an encrypted connection to the ga
teway, consider using HTTPS.

Deployed. 202 Accepted.
URL: http://34.76.216.144:8080/function/nodeinfo
```



```
user@ubuntu:~/Desktop/faas$ curl $GATEWAY_IP:8080/function/
nodeinfo
Hostname: nodeinfo-7d6db98958-6qtnq

Arch: x64
CPUs: 1
Total mem: 3680MB
Platform: linux
Uptime: 902.05
```



NodeInfo
Get info about the machine that you're deployed on. Tells CPU count, hostname, OS, and Uptime

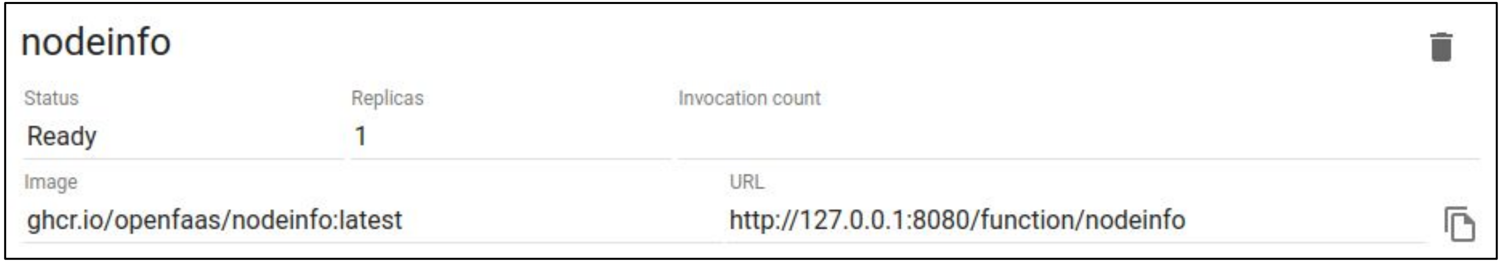
alpine
An Alpine Linux shell, set the "fprocess" to a bash built-in you want to run like "env" or "cal". By default alpine runs "cat" and can be used to echo the HTTP input.

env
Print the environment variables present in the function and HTTP request

sleep
Simulate a 2s duration or pass an X-Sleep header and a valid Golang duration

shasum
Generate a shasum for the given input

Figlet
Generate ASCII logos with the figlet CLI



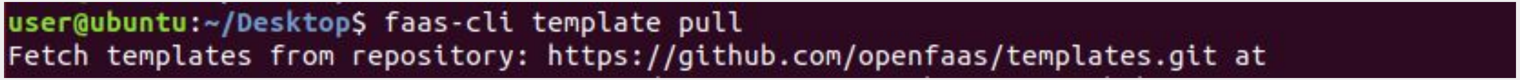
nodeinfo

Status	Replicas	Invocation count
Ready	1	

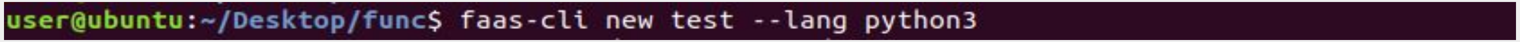
Image	URL
ghcr.io/openfaas/nodeinfo:latest	http://127.0.0.1:8080/function/nodeinfo

3. Create own function

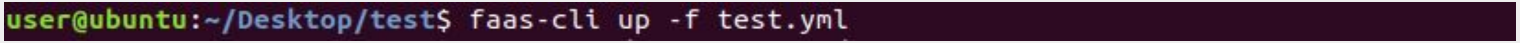
Prerequisite: Docker -> hub.docker.io account and connected to local instance

1. 

```
user@ubuntu:~/Desktop$ faas-cli template pull
Fetch templates from repository: https://github.com/openfaas/templates.git at
```

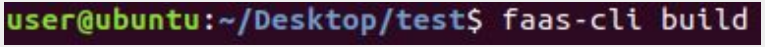
1. 

```
user@ubuntu:~/Desktop/func$ faas-cli new test --lang python3
```

1. 

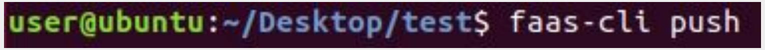
```
user@ubuntu:~/Desktop/test$ faas-cli up -f test.yml
```

up command:

1. 

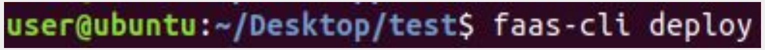
```
user@ubuntu:~/Desktop/test$ faas-cli build
```

-> build docker image

1. 

```
user@ubuntu:~/Desktop/test$ faas-cli push
```

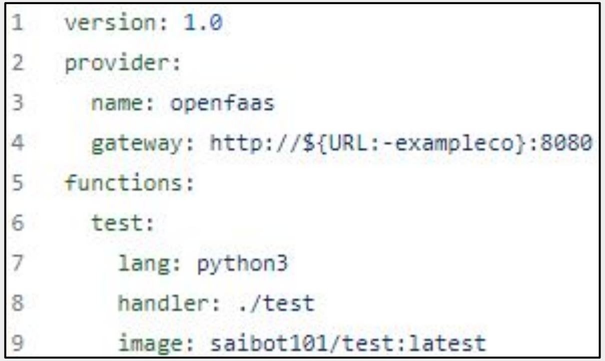
-> push docker image to local repository

1. 

```
user@ubuntu:~/Desktop/test$ faas-cli deploy
```

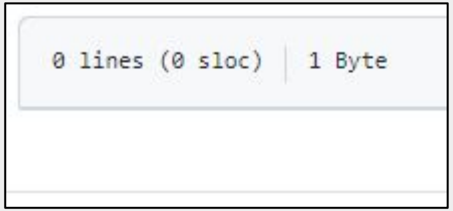
-> push docker image to public docker repository

3. Example Openfaas function



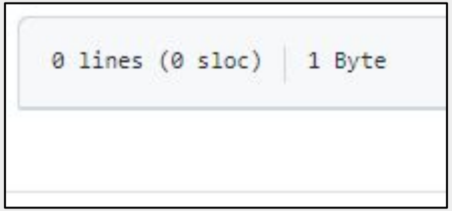
```yaml
version: 1.0
provider:
  name: openfaas
  gateway: http://${URL:-exampleco}:8080
functions:
  test:
    lang: python3
    handler: ./test
    image: saibot101/test:latest
```

test.yml



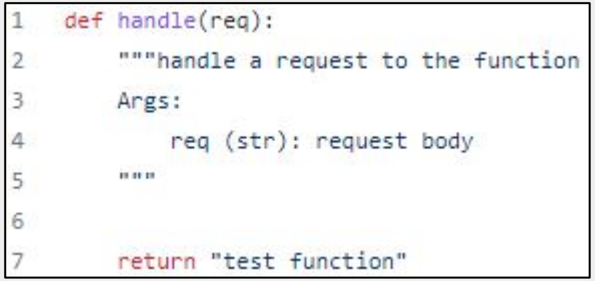
```
0 lines (0 sloc) | 1 Byte
```

__init__.py



```
0 lines (0 sloc) | 1 Byte
```

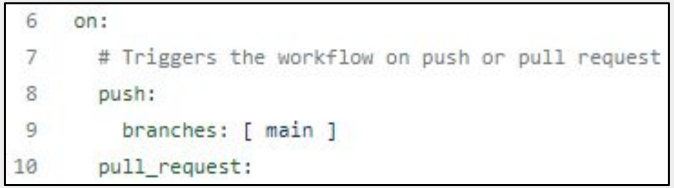
requirements.txt



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

    return "test function"
```

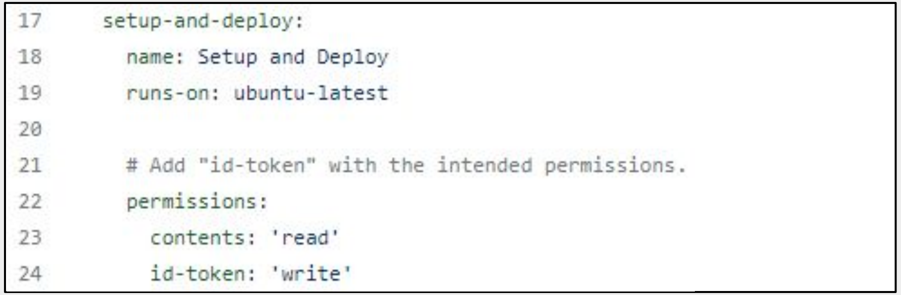
handler.py

4. Pipelines - Initialisation



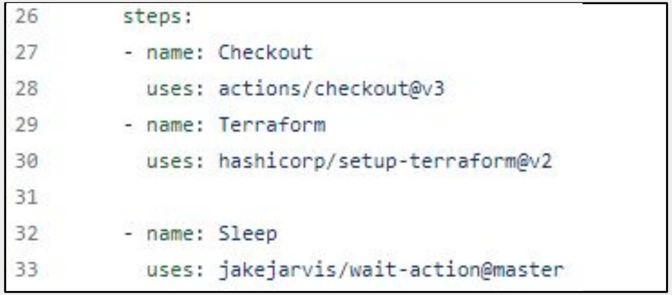
```
on:
  # Triggers the workflow on push or pull request
  push:
    branches: [ main ]
  pull_request:
```

1. Setup trigger



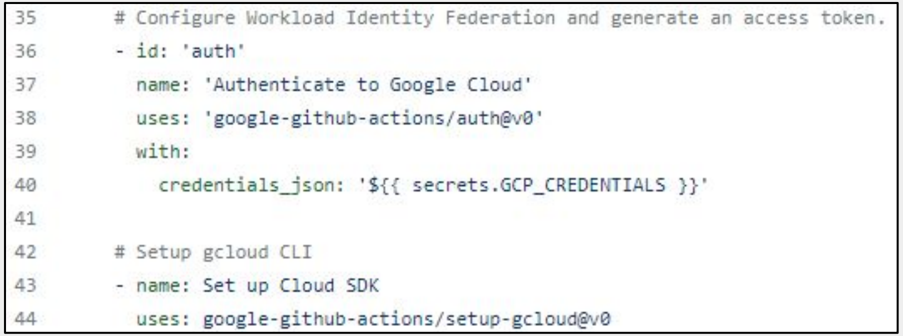
```
  setup-and-deploy:
    name: Setup and Deploy
    runs-on: ubuntu-latest

    # Add "id-token" with the intended permissions.
    permissions:
      contents: 'read'
      id-token: 'write'
```

2. Set VM and permissions



```
    steps:
    - name: Checkout
      uses: actions/checkout@v3
    - name: Terraform
      uses: hashicorp/setup-terraform@v2

    - name: Sleep
      uses: jakejarvis/wait-action@master
```

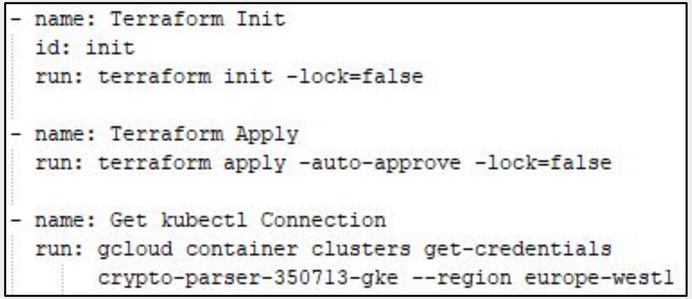
3. Load packages



```
    # Configure Workload Identity Federation and generate an access token.
    - id: 'auth'
      name: 'Authenticate to Google Cloud'
      uses: 'google-github-actions/auth@v0'
      with:
        credentials_json: '${{ secrets.GCP_CREDENTIALS }}'

    # Setup gcloud CLI
    - name: Set up Cloud SDK
      uses: google-github-actions/setup-gcloud@v0
```

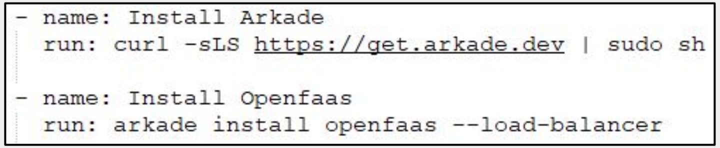
4. Setup Google Cloud

Pipeline - Create Cluster and install function from store



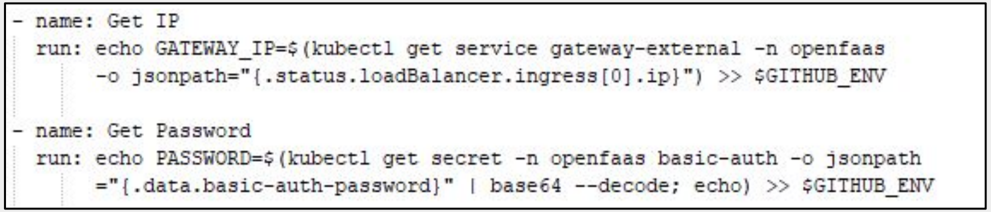
```
- name: Terraform Init
  id: init
  run: terraform init -lock=false

- name: Terraform Apply
  run: terraform apply -auto-approve -lock=false

- name: Get kubectl Connection
  run: gcloud container clusters get-credentials
        crypto-parser-350713-gke --region europe-west1
```

1. Create cluster and get credentials



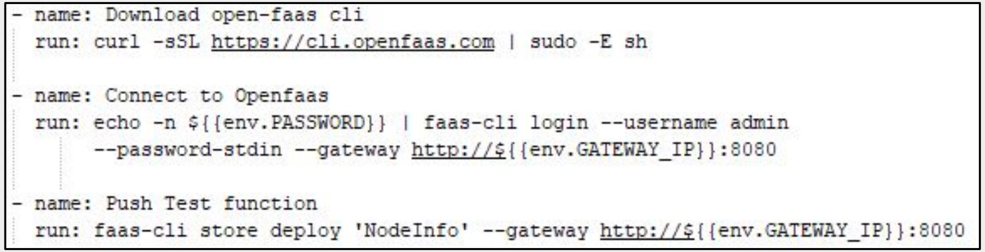
```
- name: Install Arkade
  run: curl -sLS https://get.arkade.dev | sudo sh

- name: Install Openfaas
  run: arkade install openfaas --load-balancer
```

2. Install Openfaas



```
- name: Get IP
  run: echo GATEWAY_IP=$(kubectl get service gateway-external -n openfaas
        -o jsonpath="{.status.loadBalancer.ingress[0].ip}") >> $GITHUB_ENV

- name: Get Password
  run: echo PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath
        ="{.data.basic-auth-password}" | base64 --decode; echo) >> $GITHUB_ENV
```

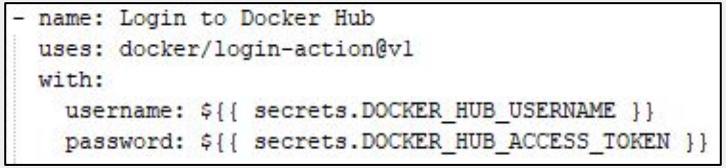
3. Get IP and password



```
- name: Download open-faas cli
  run: curl -sSL https://cli.openfaas.com | sudo -E sh

- name: Connect to Openfaas
  run: echo -n ${{env.PASSWORD}} | faas-cli login --username admin
        --password-stdin --gateway http://${{env.GATEWAY_IP}}:8080

- name: Push Test function
  run: faas-cli store deploy 'NodeInfo' --gateway http://${{env.GATEWAY_IP}}:8080
```

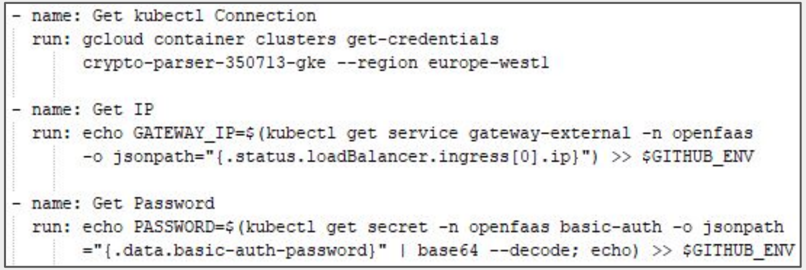
4. Login to Openfaas install function

Pipeline - Upload to existing Cluster



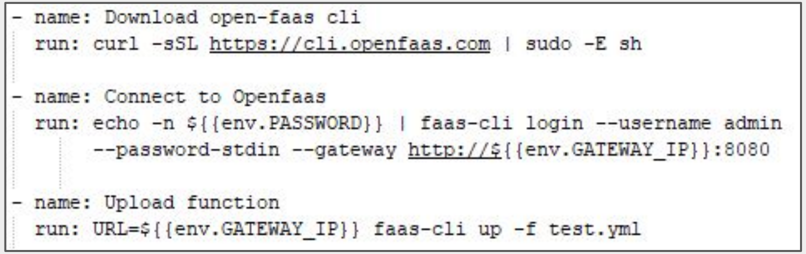
```
- name: Login to Docker Hub
  uses: docker/login-action@v1
  with:
    username: ${{ secrets.DOCKER_HUB_USERNAME }}
    password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}
```

1. Connection to Docker Hub



```
- name: Get kubectl Connection
  run: gcloud container clusters get-credentials
       crypto-parser-350713-gke --region europe-west1

- name: Get IP
  run: echo GATEWAY_IP=$(kubectl get service gateway-external -n openfaas
       -o jsonpath="{.status.loadBalancer.ingress[0].ip}") >> $GITHUB_ENV

- name: Get Password
  run: echo PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath
       ="{.data.basic-auth-password}" | base64 --decode; echo) >> $GITHUB_ENV
```

2. Get Credentials, IP and password



```
- name: Download open-faas cli
  run: curl -sSL https://cli.openfaas.com | sudo -E sh

- name: Connect to Openfaas
  run: echo -n ${{env.PASSWORD}} | faas-cli login --username admin
       --password-stdin --gateway http://${{env.GATEWAY_IP}}:8080

- name: Upload function
  run: URL=${{env.GATEWAY_IP}} faas-cli up -f test.yml
```

3. Connect to Openfaas and upload function