

R V COLLEGE OF ENGINEERING

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)



A Report On

MicroK8s with Flask

Submitted in partial fulfillment for Assignment component of

**Cloud Computing Technology
18MCN1B1**

of

MASTER OF TECHNOLOGY

in

COMPUTER NETWORK ENGINEERING

Submitted by

**AKASH HEGDE
1RV18SCN01**

Under the guidance of

Dr. B. SATHISH BABU

**PROFESSOR, DEPT. OF COMPUTER SCIENCE AND ENGINEERING
R.V. COLLEGE OF ENGINEERING**

**R V COLLEGE OF ENGINEERING
R.V.VIDYANIKETAN POST, MYSURU ROAD,
BENGALURU- 560059**

2018-19

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned the efforts with success.

It is indeed a great pleasure for me to present this self-study report on “MicroK8s with Flask” as a part of the course Cloud Computing Technology in the 1st semester of Master of Technology in Computer Network Engineering.

I would like to thank Management of R V College of Engineering for providing such a healthy environment for the successful completion of self-study work.

It gives me immense pleasure to thank Dr. Ramakanth Kumar P, Professor and Head of Department for his constant support and encouragement.

Also, I would like to express my gratitude to my course guide Dr. B. Sathish Babu, Professor, Department of Computer Science & Engineering and all other teaching and non-teaching staff of Computer Science Department who has directly or indirectly helped me in the completion of the self-study work.

Last, but not the least, I would hereby acknowledge and thank my parents who have been a source of inspiration and also instrumental in the successful completion of the self-study work.

Akash Hegde
Dept. of CNE

List of Figures

Sl.No.	Figure Name	Page No.
1	Dockerfile	5
2	requirements.txt file	5
3	test_server.py file	6
4	kubetest.yaml file	6
5	Dashboard	7
6	kubetest details	8
7	MicroK8s namespaces	8
8	Default endpoint	9
9	Login page	9
10	Logged in to the session	9
11	/health endpoint	10
12	/hello/ endpoint	10

Table of Contents

Acknowledgement

(i)

List of Figures

(ii)

Chapter 1

1. Introduction

1.1 Kubernetes

1.2 MicroK8s

1.3 Flask

1-3

Chapter 2

2. Implementation

2.1 Requirements

2.2 Installation

2.3 Configuration

2.4 Deployment

4-10

Chapter 1

Introduction

This section provides a brief introduction to Kubernetes, MicroK8s and Flask and lists their respective features.

1.1 Kubernetes

Kubernetes (K8s) is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF). It groups containers that make up an application into logical units for easy management and discovery.

Some of the features of Kubernetes are –

- Service discovery and load balancing – There is no need to modify an application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.
- Storage orchestration – Automatically mounts the storage system, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
- Automated rollouts and rollbacks – Kubernetes progressively rolls out changes to the application or its configuration, while monitoring application health to ensure it doesn't kill all the instances at the same time. If something goes wrong, Kubernetes will rollback the change.
- Batch execution – In addition to services, Kubernetes can manage batches and CI workloads, replacing containers that fail, if desired.
- Automatic binpacking – Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. It can mix

critical and best-effort workloads in order to drive up utilization and save even more resources.

- Self-healing – Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- Secret and configuration management – Deploys and updates secrets and application configuration without rebuilding the image and without exposing secrets in the stack configuration.
- Horizontal scaling – Scales an application up and down with a simple command, with a UI, or automatically based on CPU usage.

1.2 MicroK8s

MicroK8s is a small, fast, secure, single node Kubernetes that installs on just about any Linux machine. It can be used for offline development, prototyping, testing, or on a Virtual Machine as a small, cheap, reliable Kubernetes for CI/CD. It is also a great Kubernetes for appliances. IoT apps can be developed for K8s and deployed on MicroK8s on a local machine.

Some of the features of MicroK8s are –

- MicroK8s includes a docker registry, so it is possible to make containers, push them, and deploy them all on a single laptop.
- It runs safely with state-of-the-art isolation.
- CNCF binaries are delivered automatically, with updates and upgrades.
- When a new major version comes out, it can be upgraded with a single command (or automatically).
- A GPGPU and docker containers can be used to run CUDA applications.
- It can also be used in CI/CD pipelines of DevOps.

1.3 Flask

Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Some of the features of Flask are –

- It contains a development server and debugger.
- Flask provides integrated support for unit testing.
- It can be used for RESTful request dispatching.
- It uses Jinja2 templating.
- Flask provides support for secure cookies (client side sessions).
- It is WSGI 1.0 compliant.
- It is Unicode-based.
- It has extensive documentation.
- Flask also provides Google App Engine compatibility.
- There are many extensions available to enhance features desired.

Chapter 2

Implementation

This section describes the implementation of MicroK8s on a local machine. It also describes the deployment of a Flask application on containers pushed by MicroK8s.

2.1 Requirements

The machine that is being used to deploy MicroK8s needs to have the following hardware and software requirements –

- A 64-bit dual-core processor.
- At least 2GB of RAM.
- Ubuntu 16.04 LTS or higher.
- Optional GPU to support CUDA operations.

2.2 Installation

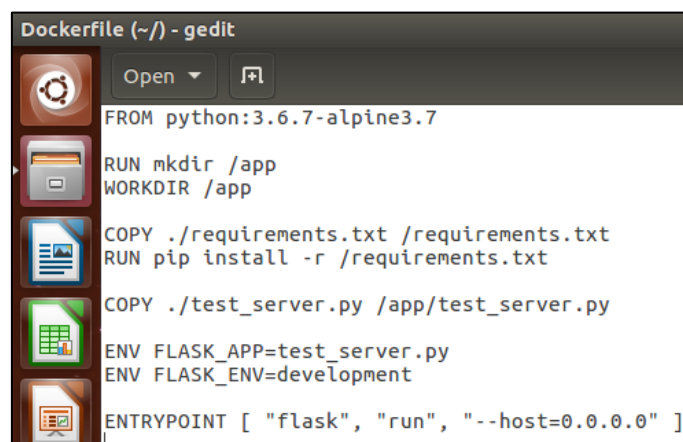
There are some application software that have to be installed in order to carry out this implementation.

- Snap can be installed using the following command –
`sudo apt-get install snapd`
- MicroK8s can be installed using snap as follows –
`sudo snap install microk8s --classic`
- Python3 and pip3 can be installed using the following command –
`sudo apt-get install python3-pip`
- Flask can be installed using pip3 as follows –
`pip3 install flask`

2.3 Configuration

Once the required application software are installed, some files have to be created and configured in order to do the implementation in the next stage.

- A Dockerfile has to be written in order to specify the environment that the image of the Flask application has to be containerized on. A requirements file is also necessary to check if the required software have been installed on the machine.

A screenshot of a gedit window titled "Dockerfile (~/) - gedit". The window shows a Dockerfile with the following content:

```
FROM python:3.6.7-alpine3.7

RUN mkdir /app
WORKDIR /app

COPY ./requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

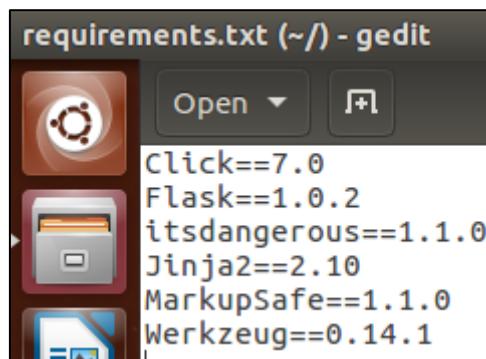
COPY ./test_server.py /app/test_server.py

ENV FLASK_APP=test_server.py
ENV FLASK_ENV=development

ENTRYPOINT [ "flask", "run", "--host=0.0.0.0" ]
```

The left sidebar of the gedit window shows icons for Docker, file explorer, and other standard editors.

Figure 2.1 Dockerfile

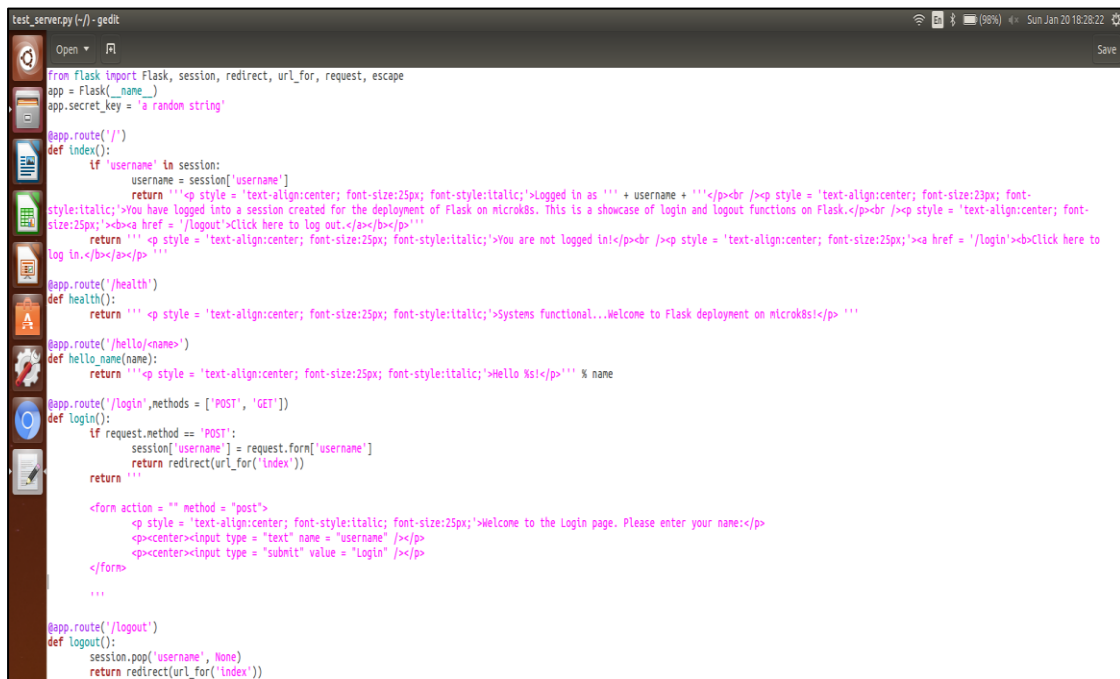
A screenshot of a gedit window titled "requirements.txt (~/) - gedit". The window shows a requirements.txt file with the following content:

```
Click==7.0
Flask==1.0.2
itsdangerous==1.1.0
Jinja2==2.10
MarkupSafe==1.1.0
Werkzeug==0.14.1
```

The left sidebar of the gedit window shows icons for Docker, file explorer, and other standard editors.

Figure 2.2 requirements.txt file

- The application to be deployed on the local cloud is written in Python.



```
test_server.py (-/) - gedit
from flask import Flask, session, redirect, url_for, request, escape
app = Flask(__name__)
app.secret_key = 'a random string'

@app.route('/')
def index():
    if 'username' in session:
        username = session['username']
        return '''<p style = 'text-align:center; font-size:25px; font-style:italic;'>Logged in as ''' + username + '''</p><br /><p style = 'text-align:center; font-size:23px; font-style:italic;'>You have logged into a session created for the deployment of Flask on microk8s. This is a showcase of login and logout functions on Flask.</p><br /><p style = 'text-align:center; font-size:25px;'><a href = '/logout'>Click here to log out.</a></b></p>'''
    return '''<p style = 'text-align:center; font-size:25px; font-style:italic;'>You are not logged in</p><br /><p style = 'text-align:center; font-size:25px;'><a href = '/login'>Click here to log in.</a></b></p>'''

@app.route('/health')
def health():
    return '''<p style = 'text-align:center; font-size:25px; font-style:italic;'>Systems functional...Welcome to Flask deployment on microk8s!</p>'''

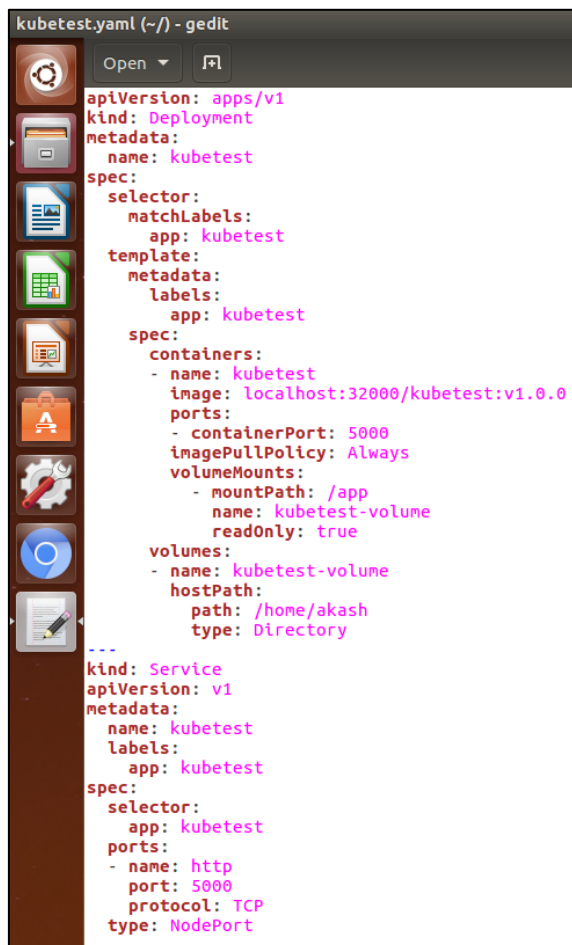
@app.route('/hello/<name>')
def hello_name(name):
    return '''<p style = 'text-align:center; font-size:25px; font-style:italic;'>Hello <name>!</p>''' % name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
    <form action = "" method = "post">
    <p style = 'text-align:center; font-style:italic; font-size:25px;'>Welcome to the Login page. Please enter your name:</p>
    <p><center><input type = "text" name = "username" /></p>
    <p><center><input type = "submit" value = "Login" /></p>
    </form>
    '''

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))
```

Figure 2.3 test_server.py file

- Finally, the configurations for the endpoints are specified in a YAML file.



```
kubetest.yaml (-/) - gedit
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubetest
spec:
  selector:
    matchLabels:
      app: kubetest
  template:
    metadata:
      labels:
        app: kubetest
    spec:
      containers:
        - name: kubetest
          image: localhost:32000/kubetest:v1.0.0
          ports:
            - containerPort: 5000
          imagePullPolicy: Always
          volumeMounts:
            - mountPath: /app
              name: kubetest-volume
              readOnly: true
      volumes:
        - name: kubetest-volume
          hostPath:
            path: /home/akash
            type: Directory
---
kind: Service
apiVersion: v1
metadata:
  name: kubetest
  labels:
    app: kubetest
spec:
  selector:
    app: kubetest
  ports:
    - name: http
      port: 5000
      protocol: TCP
  type: NodePort
```

Figure 2.4 kubetest.yaml file

2.4 Deployment

Deployment of the Flask application on MicroK8s goes through some steps. These are detailed as follows –

- Enable the registry –
`microk8s.enable registry`
- Enable the dashboard –
`microk8s.enable dashboard`
- Test the app on Flask –
`FLASK_APP=test_server.py flask run`
- Check `http://localhost:5000` to see the results.
- Build the image on Docker –
`microk8s.docker build -t localhost:32000/kubetest:v1.0.0 .`
- Push the image on to Docker –
`microk8s.docker push localhost:32000/kubetest:v1.0.0`
- Set up the cluster –
`microk8s.kubectl apply -f kubetest.yaml`
- Set up a proxy server to run on the localhost –
`microk8s.kubectl proxy`
- Go to the following link to see the dashboard –
`http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/`

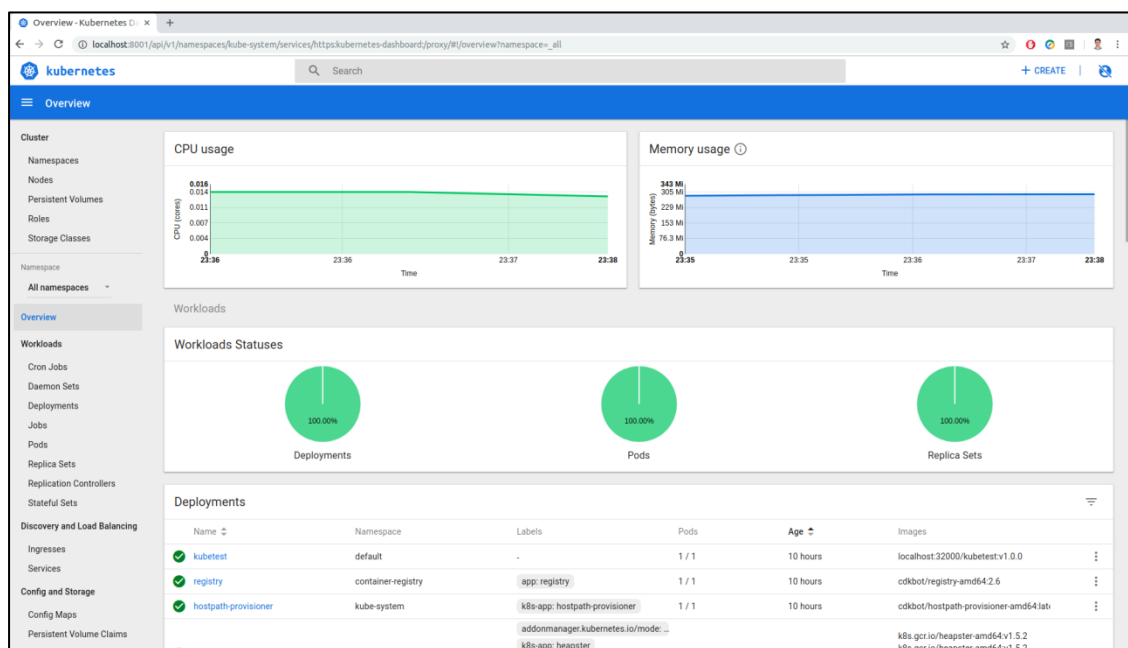


Figure 2.5 Dashboard

- The kubectl deployment can be seen as –

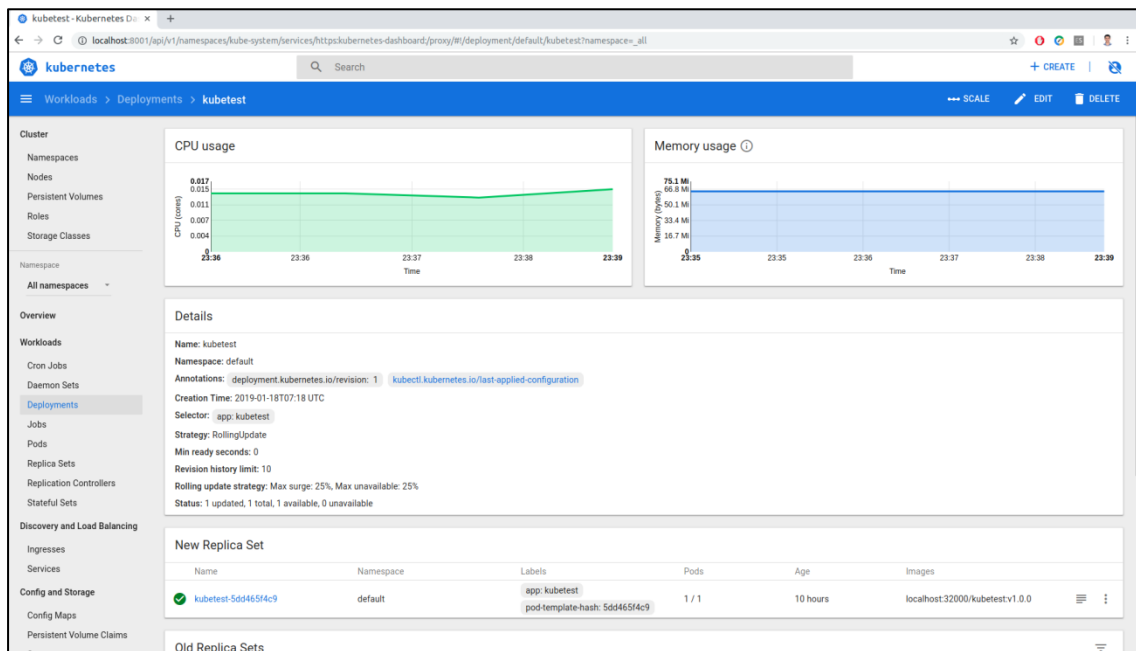


Figure 2.6 kubetest details

- The services and deployments can also be viewed in the terminal –

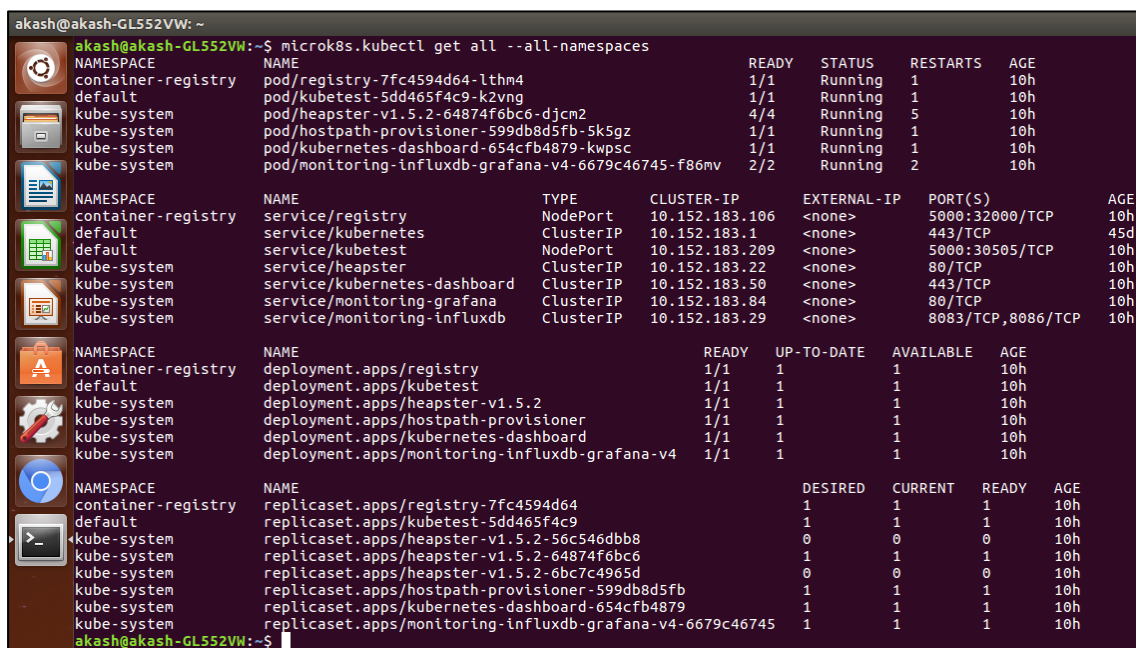


Figure 2.7 MicroK8s namespaces

- The deployment of Flask can be observed on <http://localhost:5000> and the alternate endpoints `/hello/` and `/health`. A session is created and the user can observe the login and logout features of the session.

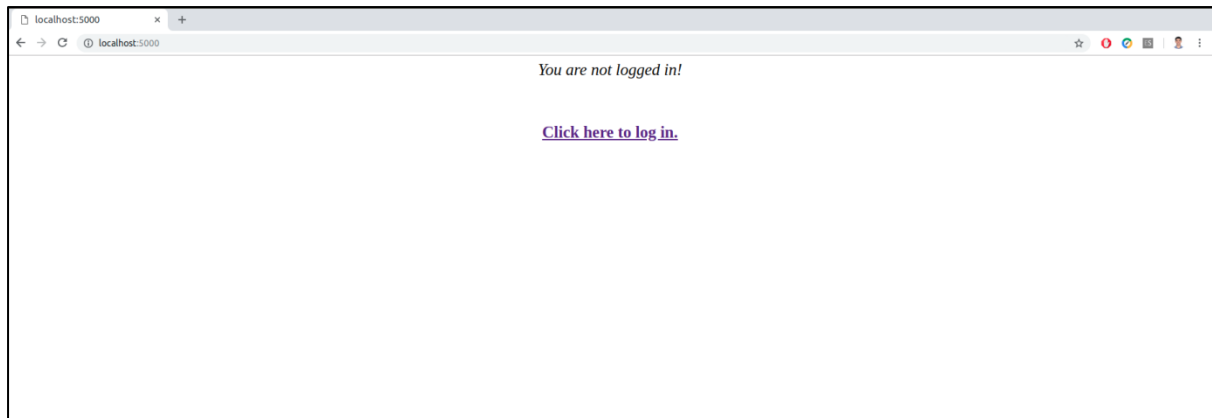


Figure 2.8 Default endpoint

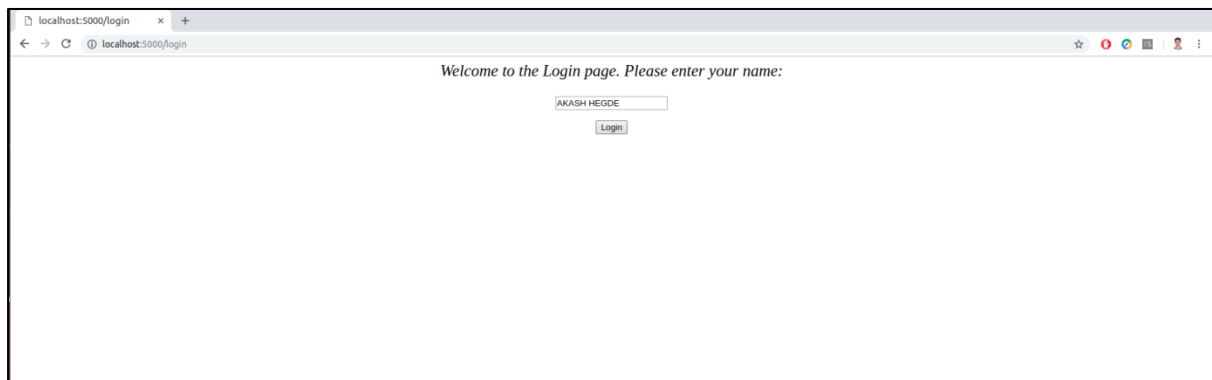


Figure 2.9 Login page

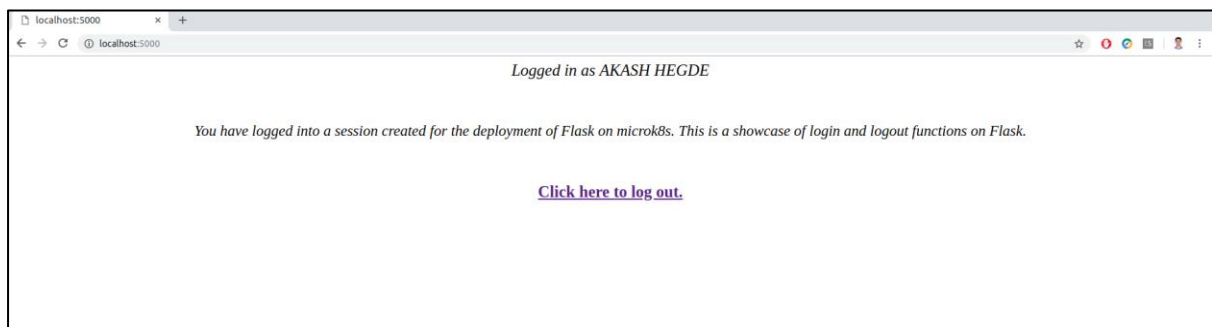


Figure 2.10 Logged in to the session

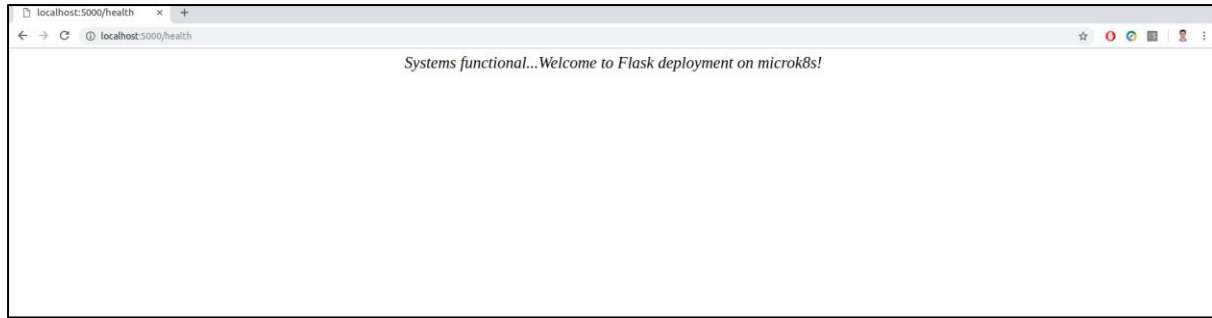


Figure 2.11 /health endpoint

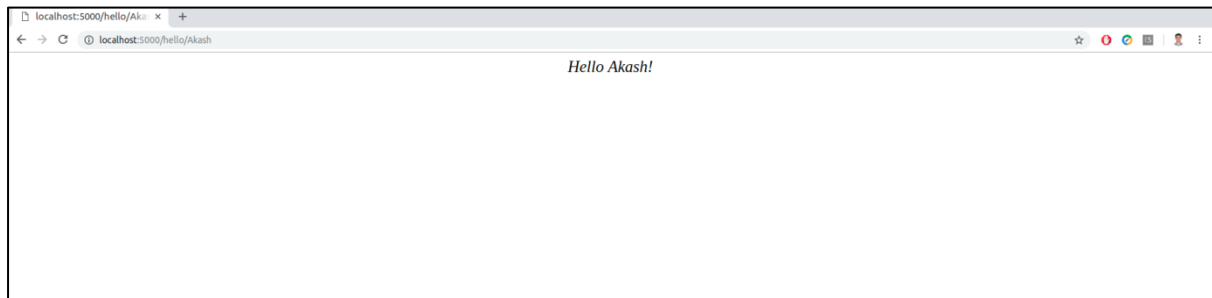


Figure 2.12 /hello/ endpoint