

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

(Online Organic Store Management)

Domain: Cloud Application Development

Team Id: PNT2022MID10625

Team Leader: Arularasi. J – 191021013

Team Members:

Abinaya. D – 191021002

Abitha. N – 191021005

Dhanya Lakshmi. P – 191021022

Ezhilmathy. E - 191021027

DEPLOYMENT OF APP IN IBM CLOUD:

STEP 1: Containerize the App

STEP 2: Upload the image to IBM Registry

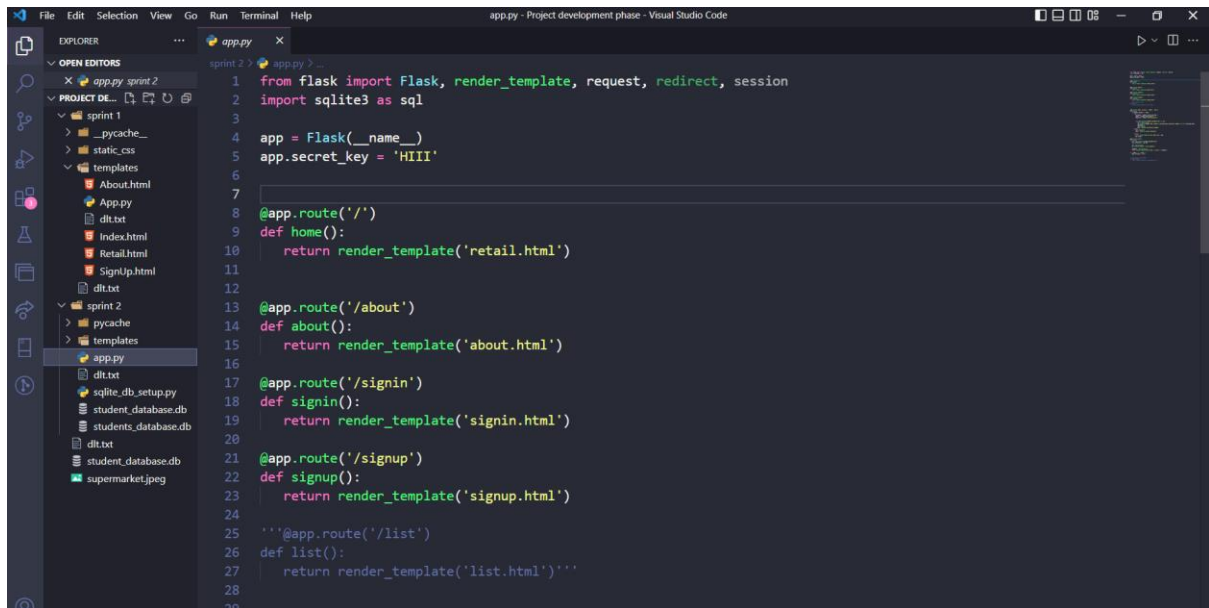
STEP 3: Deploy in Kubernetes Cluster

STEP 1 : CONTAINERIZE THE APP:

Docker Image Creation

Make Project Folder

Open your terminal and make a folder for your flask application



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file explorer shows a project structure with folders for 'sprint 1' and 'sprint 2', and files for 'App.py', 'dlt.txt', 'Index.html', 'Retail.html', 'SignUp.html', 'student_database.db', and 'supermarket.jpeg'. The code editor shows the following Python code:

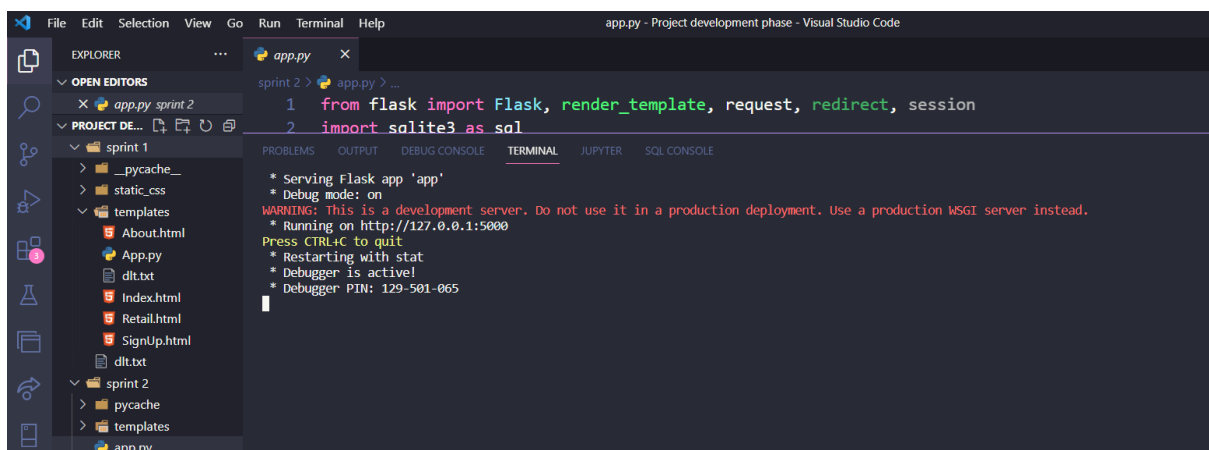
```
1 from flask import Flask, render_template, request, redirect, session
2 import sqlite3 as sql
3
4 app = Flask(__name__)
5 app.secret_key = 'HIII'
6
7
8 @app.route('/')
9 def home():
10     return render_template('retail.html')
11
12
13 @app.route('/about')
14 def about():
15     return render_template('about.html')
16
17
18 @app.route('/signin')
19 def signin():
20     return render_template('signin.html')
21
22
23 @app.route('/signup')
24 def signup():
25     return render_template('signup.html')
26
27
28 '''@app.route('/list')
29 def list():
30     return render_template('list.html')'''
```

Insert the code to the Dockerfile:

```
FROM python:alpine3.7
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5001
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
```

Test the Flask App:

It should start our development server which comes with the flask on “http://0.0.0.0:5001/”.



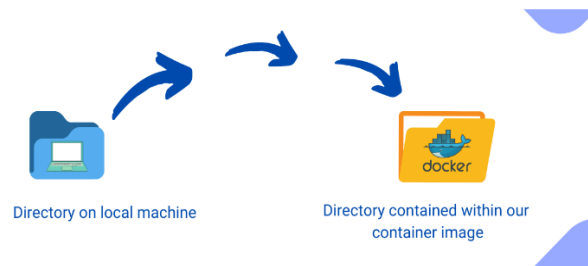
The screenshot shows the Visual Studio Code editor with the file explorer on the left and the terminal in the center. The terminal output shows the following messages:

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 129-501-065
```

Creating Docker image of the project:

`sudo docker build --tag flask-docker-demo-app .`

The above command will create an app with the tag flask-docker-demo-app.



```
hiteshchoudhary~$docker build -t Hiteshchoudhary/hey-python-flask:0.0.1.RELEASE .
[+] Building 7.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 161B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3-alpine3.15           3.0s
=> [auth] library/python:pull token for registry-1.docker.io                   0.0s
=> [1/4] FROM docker.io/library/python:3-alpine3.15@sha256:d89ac9cefd2213b99a792fd8ec4f15c0297d9340a688b 0.0s
hiteshchoudhary~$docker container run -d -p 3000:3000 hiteshchoudhary/hey-python-flask:0.
Unable to find image 'hiteshchoudhary/hey-python-flask:0.' locally
^C
hiteshchoudhary~$docker container run -d -p 3000:3000 hiteshchoudhary/hey-python-flask:0.0.1.RELEASE
hiteshchoudhary~$
```

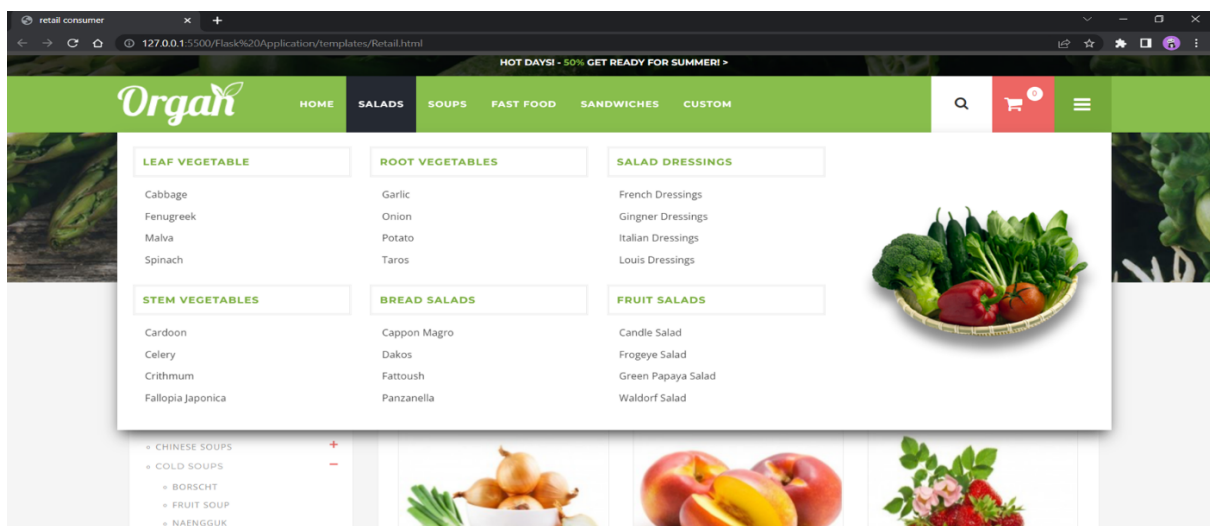
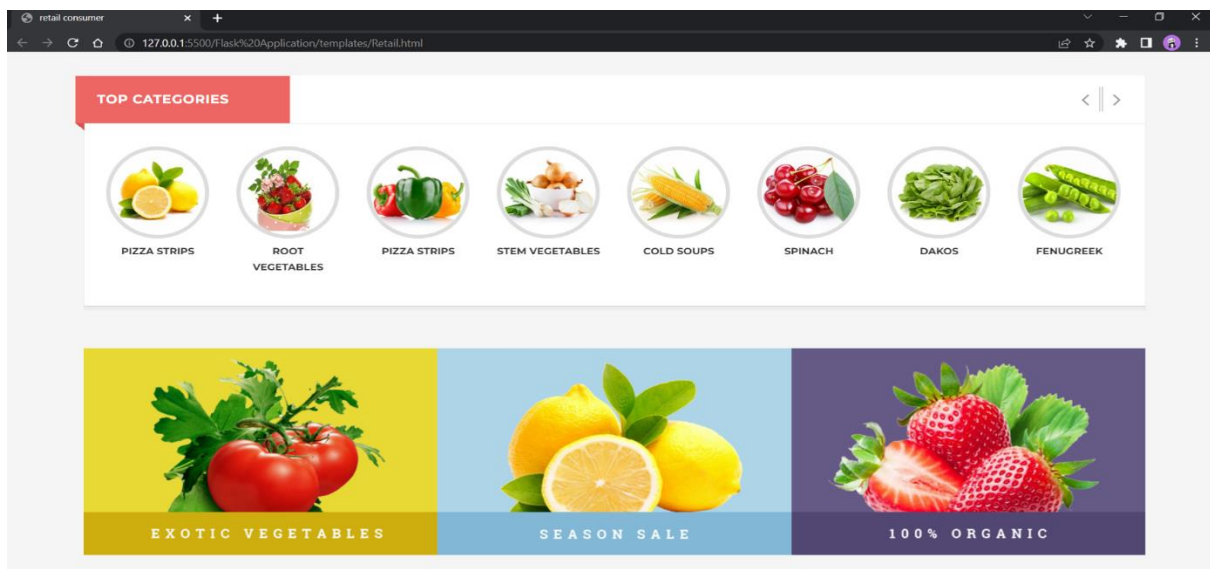
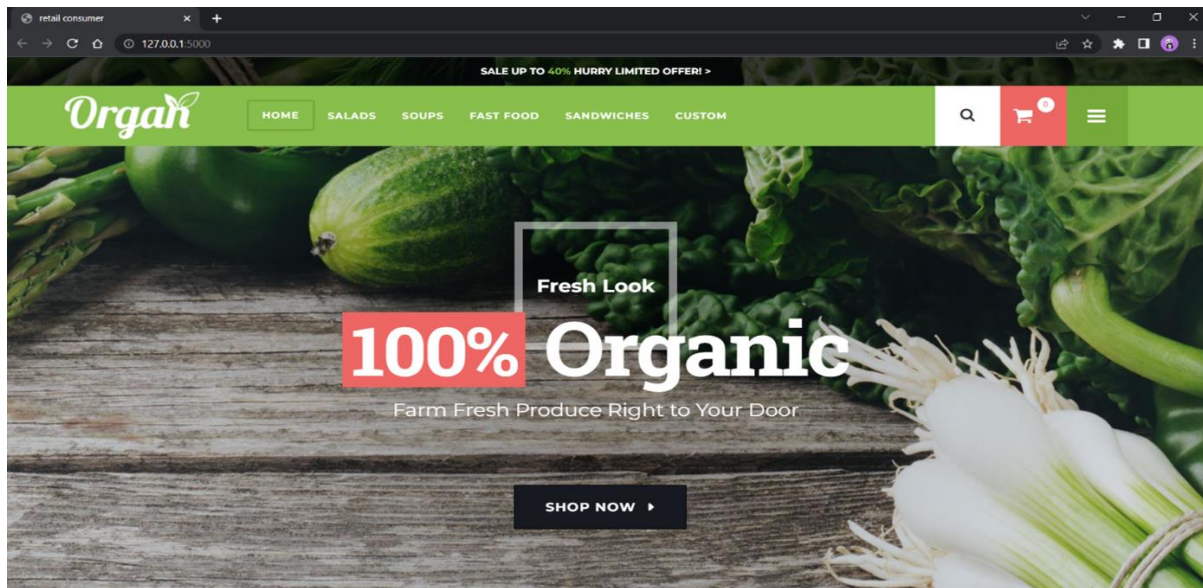
Run the docker image:

`sudo docker run --name flask-docker-demo-app -p 5001:5001 flask-docker-demo-app`

```
hiteshchoudhary~$docker container run -d -p 3000:3000 hiteshchoudhary/hey-python-flask:0.
Unable to find image 'hiteshchoudhary/hey-python-flask:0.' locally
^C
hiteshchoudhary~$docker container run -d -p 3000:3000 hiteshchoudhary/hey-python-flask:0.0.1.RELEASE

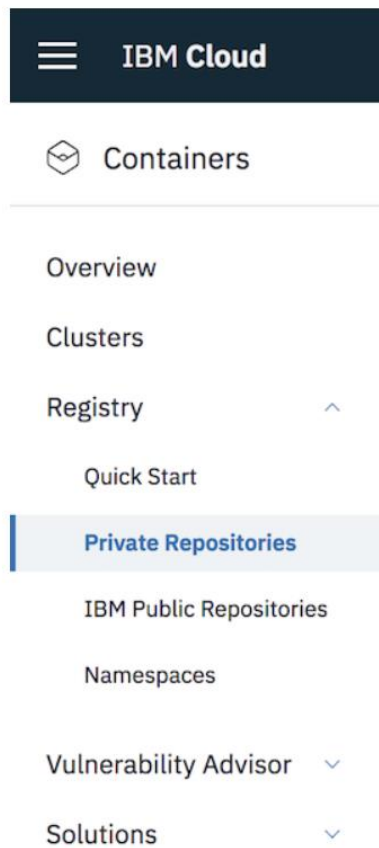
hiteshchoudhary~$docker container run -d -p 3000:3000 hiteshchoudhary/hey-python-flask:0.0.1.RELEASE
076f9da1074488b9cd7a76ca96cafb44fbb1e43b8ac6a9832ccba38aad74c7d
hiteshchoudhary~$
```

Test the App:



STEP 2 : **UPLOAD THE IMAGE TO IBM CONTAINER REGISTRY:**

1. From your account dashboard, go to IBM Cloud Kubernetes Service.
2. From the left navigation menu, select Private Repositories.



Install the Container Registry plug-in:

```
ibmcloud plugin install container-registry -r "IBM Cloud"
```

Log in to your IBM Cloud account:

```
ibmcloud login -a <cloud_foundary_end_point_for_the_region>
```

Name and create your namespace. Use this namespace for the rest of the Quick Start.

```
ibmcloud cr namespace-add <namespace>
```

Log your local Docker daemon into the IBM Cloud Container Registry.

```
ibmcloud cr login
```

Push the image:

```
kunals-mbp:web kunalmalhotra$ docker push registry.ng.bluemix.net/flask-node/app:latest
The push refers to repository [registry.ng.bluemix.net/flask-node/app]
d905410b27c1: Pushed
b96dea950728: Pushed
437e8db4a234: Pushed
ba9884d50644: Pushed
1989aa0f3739: Layer already exists
7be9e9e9c283: Layer already exists
1172bcd1177f: Layer already exists
8eb4c3e69e64: Layer already exists
1fa8778eb779: Layer already exists
fa0c3f992cbd: Layer already exists
ce6466f43b11: Layer already exists
719d45669b35: Layer already exists
3b10514a95be: Layer already exists
latest: digest: sha256:5015254c21592b5ab08168707b74ddd763e97e80b59d9187afa2a80433b9d2ab size: 3061
kunals-mbp:web kunalmalhotra$
```

Verify that your image is in your private registry:

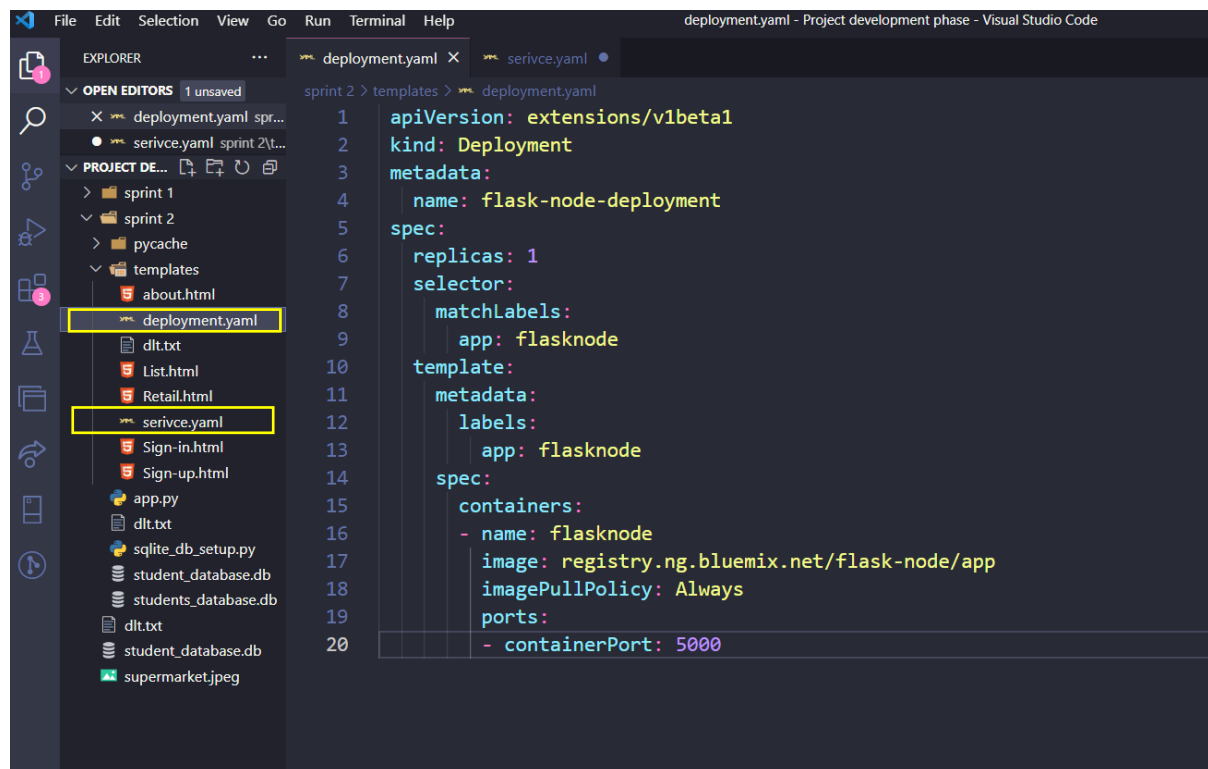
```
ibmcloud cr image-list
```

```
kunals-mbp:web kunalmalhotra$ ibmcloud cr image-list
Listing images...

REPOSITORY                                TAG      DIGEST      NAMESPACE  CREATED    SIZE    SECURITY STATUS
registry.ng.bluemix.net/flask-node/app    latest   b721dd768fe0 flask-node   1 day ago  366 MB  3 Issues
OK
kunals-mbp:web kunalmalhotra$
```

STEP 3: Deploy in Kubernetes Cluster

Create Configuration files for Kubernetes



Deploy Application to Kubernetes:

```
ibmcloud cs region-set us-south
ibmcloud cs cluster-config cluster_kunal
```


Set the KUBECONFIG environment variable. Copy the output from the previous command and paste it in your terminal.

```
> export KUBECONFIG=/Users/$USER/.bluemix/plugins/container-service/clusters/< cluster_name >/< cluster_configuration_file.yaml>
```

Verify that you can connect to your cluster by listing your worker nodes:

```
kubectl get nodes
kubectl create -f deployment.yaml
kubectl create -f service.yaml
```

Look at the Kubernetes dashboard from the IBM Kubernetes Service overview page:

The screenshot shows the IBM Kubernetes Service dashboard. The left sidebar contains navigation links for Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (default), Overview, Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), Discovery and Load Balancing (Ingresses, Services), and Config and Storage (Config Maps, Persistent Volumes, StorageClasses). The main content area displays several tables:

- Deployments:** A table with columns Name, Labels, Pods, Age, and Images. It shows one deployment: 'flask-node-deployment' with 1/1 pods, 5 minutes age, and image 'registry.ng.bluemix.net/flask-node/app'.
- Pods:** A table with columns Name, Node, Status, Restarts, Age, CPU (cores), and Memory (bytes). It shows one pod: 'flask-node-deployment-5cd96cf5bc-d6nfx' on node '10.47.79.201' in 'Running' status, 0 restarts, 5 minutes age, 0 CPU cores, and 19.352 Mi memory.
- Replica Sets:** A table with columns Name, Labels, Pods, Age, and Images. It shows one replica set: 'flask-node-deployment-5cd96cf5bc' with 1/1 pods, 5 minutes age, and image 'registry.ng.bluemix.net/flask-node/app'.
- Services:** A table with columns Name, Labels, Cluster IP, Internal endpoints, External endpoints, and Age. It shows two services: 'kubernetes' (component: apiserver, provider: kubernetes, cluster IP: 172.21.0.1) and 'flask-node-deployment' (cluster IP: 172.21.104.14).

Finally, go to your browser and ping the Public IP of your worker node:

