LAB UI MATERIAL
DOCKER AND OPENSHIFT

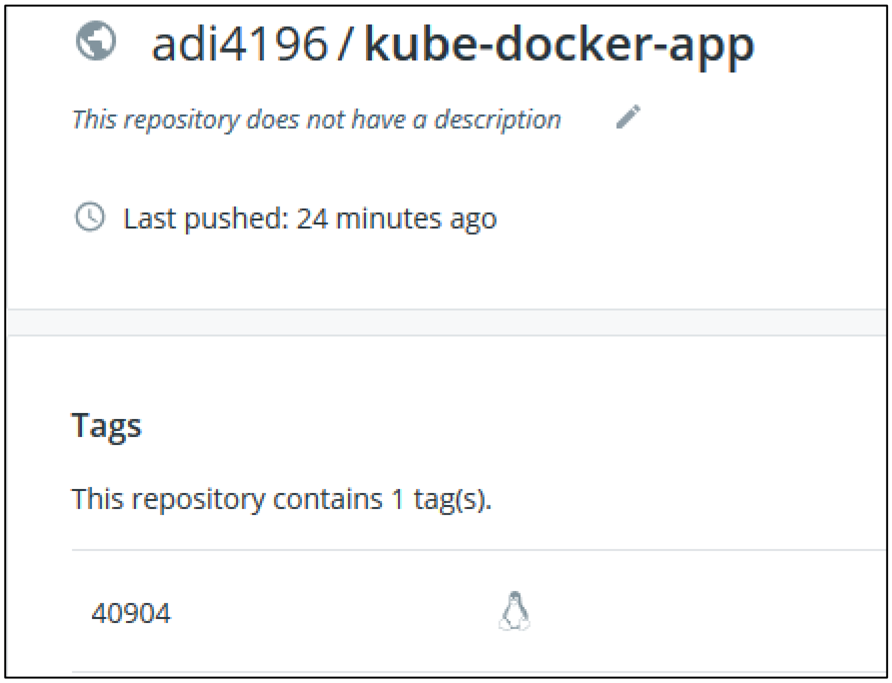
1. DOCKER PLAYGROUND

Use the below link for our preconfigured Docker instance
https://labs.play-with-docker.com

a. **To List all the Images:-**
docker images

b. **Pull our docker project from github and build the docker image:-**
git clone https://github.com/aditya4196/kube-docker-demo.git
cd kube-docker-demo
docker build –tag kube-docker-app:[tagname]
(To keep the tagname unique for yourself, use your empid as the tagname,
Example: kube-docker-app:40904)

c. **To push the built docker image to DockerHub (public docker image repository):-**
docker login –u adi4196
docker tag kube-docker-app:40904 adi4196/kube-docker-app:40904
docker push adi4196/kube-docker-app:40904


adi4196 / kube-docker-app

This repository does not have a description

Last pushed: 24 minutes ago

Tags

This repository contains 1 tag(s).

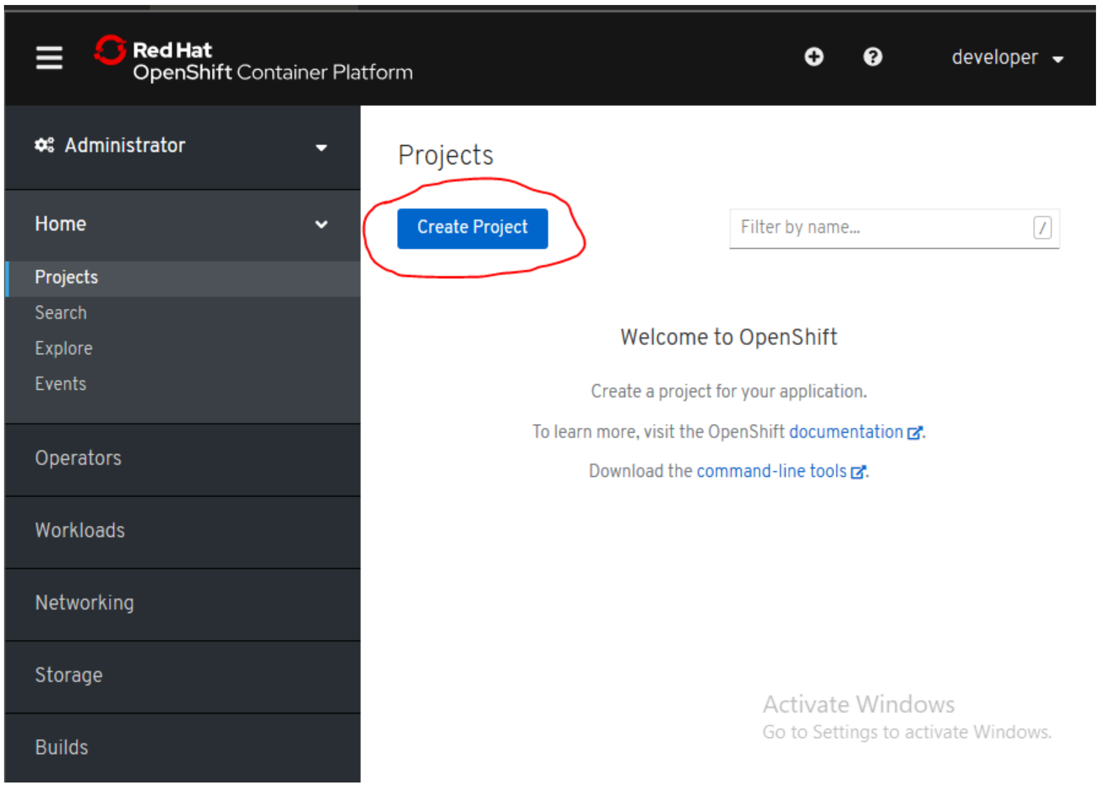
40904

2. OPENSHIFT BASIC FEATURES

Use the below link for the instance with preconfigured Openshift environment
https://www.openshift.com/learn/courses/playground

a. **Steps to Login and create Namespace (project) in Openshift:**
Go to Console
Login using username – developer, password – developer

Click on **Create Project**, assign a **Name** (eg :- demo-project) and click on **Create**



Projects

Welcome to OpenShift

Create a project for your application.

To learn more, visit the OpenShift documentation.

Download the command-line tools.

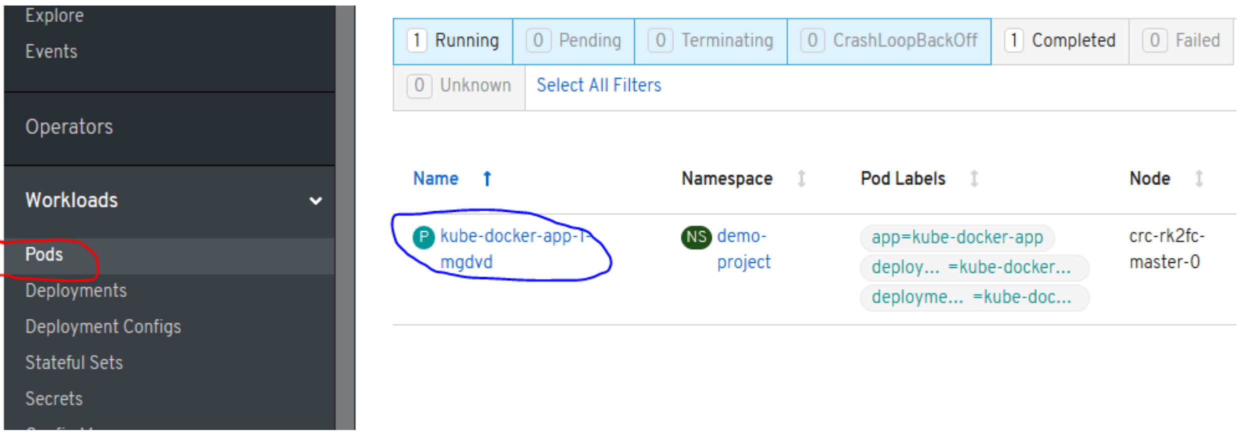
Activate Windows
Go to Settings to activate Windows.

b. **Pull the image from DockerHub and deploy in openshift**
oc new-app adi4196/kube-docker-app:40904 (My Image which pushed in DockerHub)

(Note:- We can do the above step using UI but this one command reduces our work to deploy the image and create a service in just one command hence we will be using command line (CLI) only for this step)

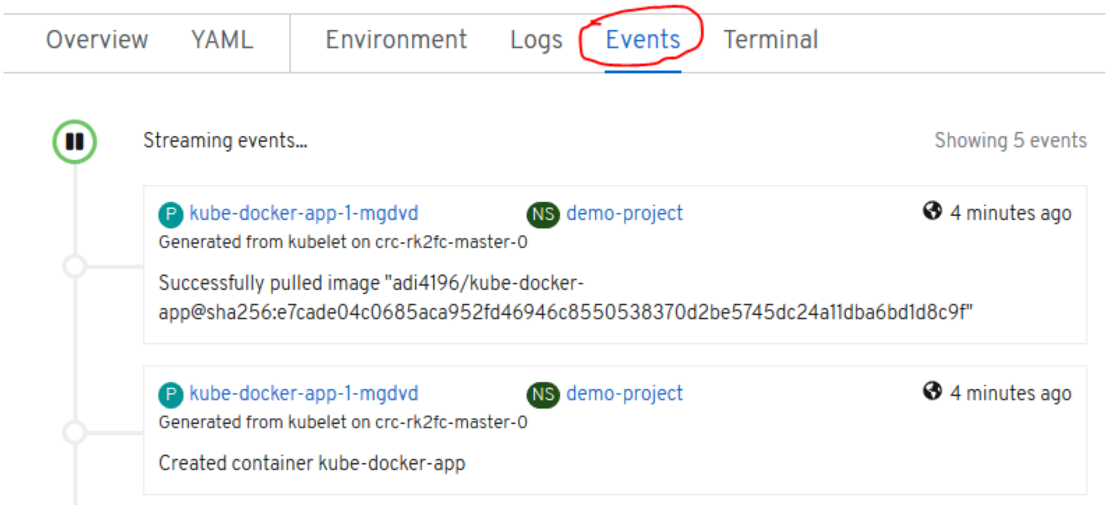
c. **Check the events of the running deployment (Process of deployment)**
Go to Workloads -→ Pods

Click on the Pod name as shown below



Name	Namespace	Pod Labels	Node
P kube-docker-app-1-mgdvd	NS demo-project	app=kube-docker-app deploy... =kube-docker... deployme... =kube-doc...	crc-rk2fc-master-0

Click on the Events Section at the top as shown below



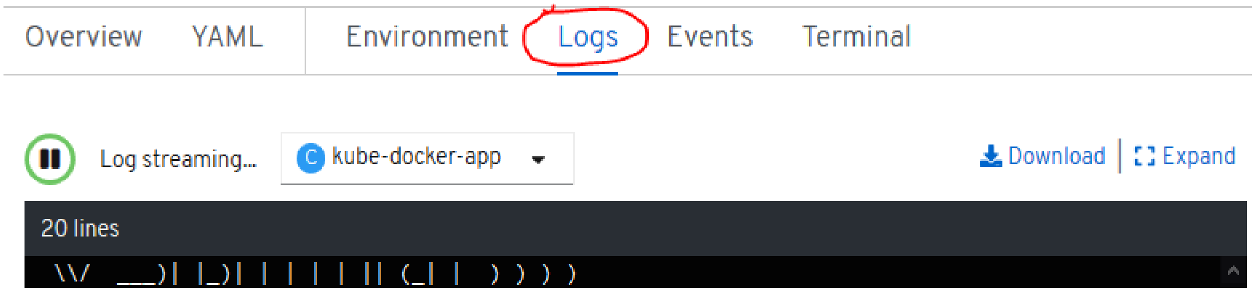
P kube-docker-app-1-mgdvd NS demo-project 4 minutes ago
Generated from kubelet on crc-rk2fc-master-0
Successfully pulled image "adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c8550538370d2be5745dc24a11dba6bd1d8c9f"

P kube-docker-app-1-mgdvd NS demo-project 4 minutes ago
Generated from kubelet on crc-rk2fc-master-0
Created container kube-docker-app

d. **Check the logs of your application running in the pod**
Go to Workloads -→ Pods

Click on the Pod name as shown below

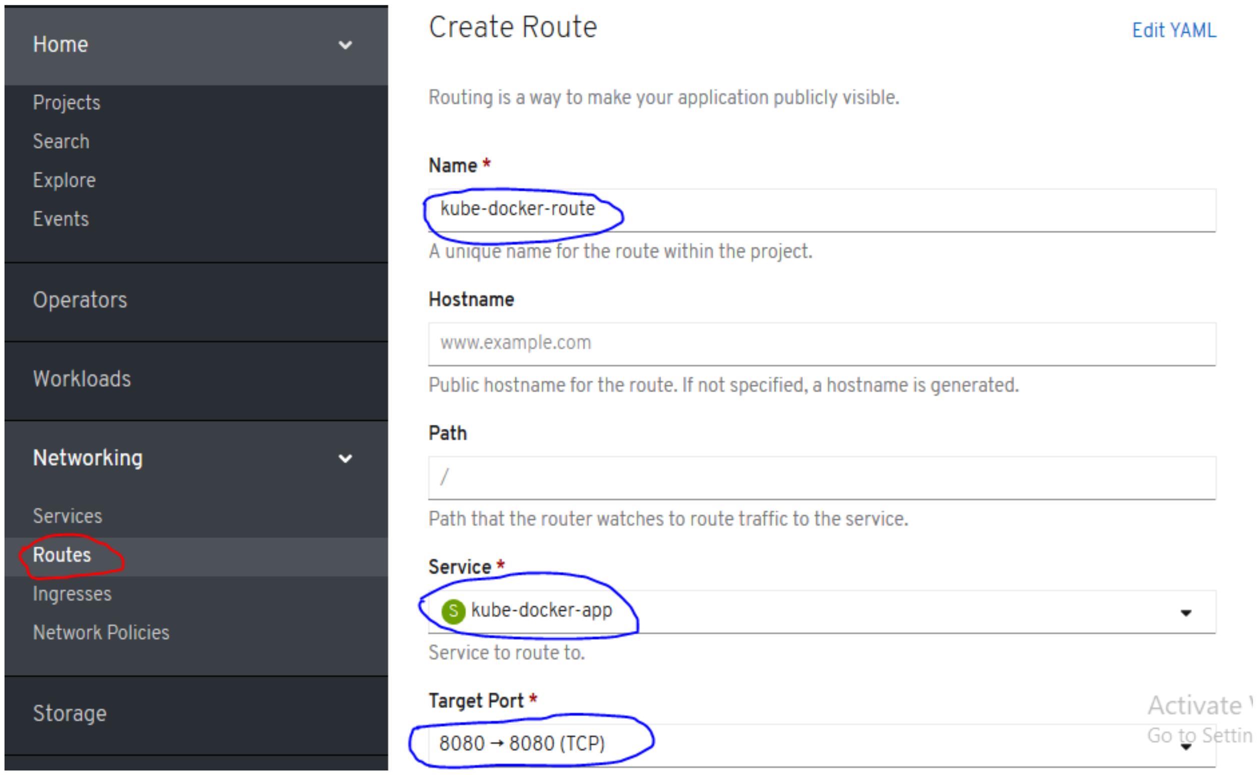
Click on the Events Section at the top as shown below



```
20 lines
 \V/  ___)| |_)| | | | | || (_| |  ) ) ) )
```

e. **Hit the service exposed for your application using Curl command or using the browser**
Go to Networking → Routes

Click on Create Route, Give the Name, Select the Service created, Assign port 8080 -> 8080(TCP) as shown below



Create Route

Routing is a way to make your application publicly visible.

Name *
kube-docker-route
A unique name for the route within the project.

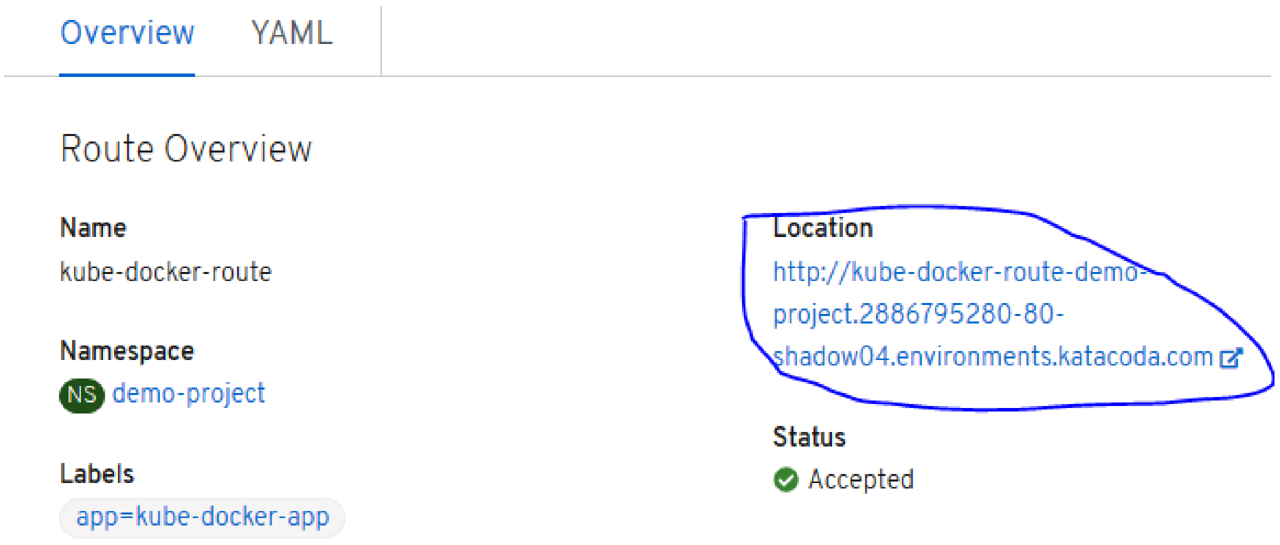
Hostname
www.example.com
Public hostname for the route. If not specified, a hostname is generated.

Path
/
Path that the router watches to route traffic to the service.

Service *
kube-docker-app
Service to route to.

Target Port *
8080 → 8080 (TCP)

Click on the below hostname which will take you to the browser



Route Overview

Name
kube-docker-route

Namespace
NS demo-project

Labels
app=kube-docker-app

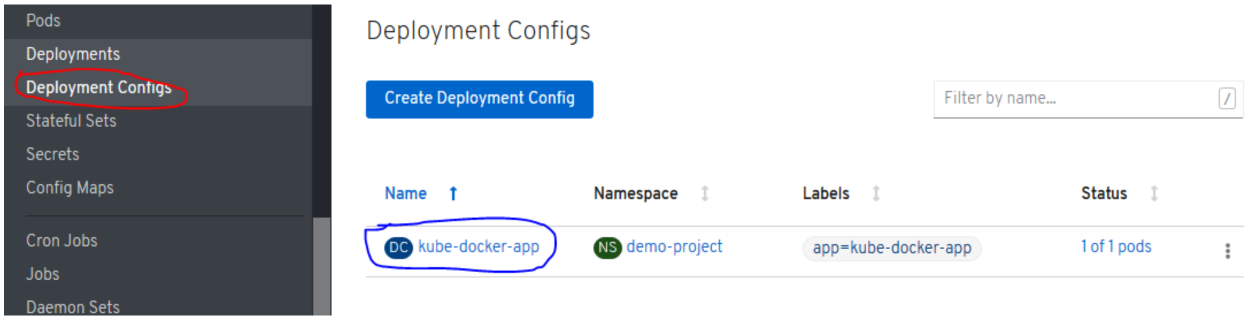
Location
http://kube-docker-route-demo-project.2886795280-80-shadow04.environments.katacoda.com

Status
Accepted

3. OPENSHIFT ADVANCED FEATURES

a. **Roll-up or Roll-down a pod of the deployed application**
Go to Workloads → Deployment Configs

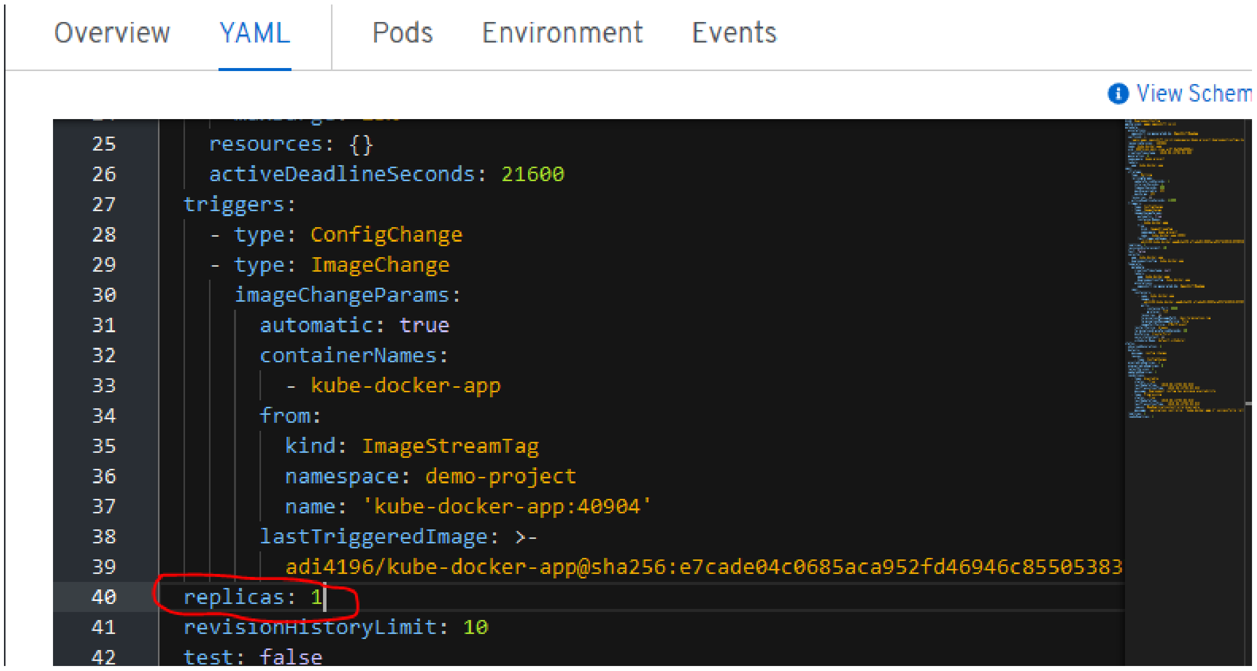
Click on the Name **'kube-docker-app'** shown below



Name	Namespace	Labels	Status
DC kube-docker-app	NS demo-project	app=kube-docker-app	1 of 1 pods

Go to YAML section and update the 'replicas' value from 1 to 2 in the editor section as shown below

Click on Save and Reload and then Go to Workloads → Pods.



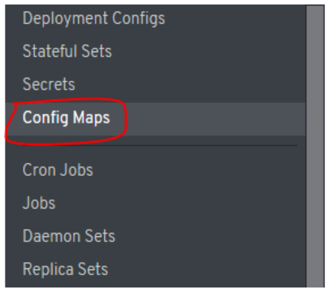
```
    resources: {}
    activeDeadlineSeconds: 21600
  triggers:
    - type: ConfigChange
    - type: ImageChange
      imageChangeParams:
        automatic: true
        containerNames:
          - kube-docker-app
        from:
          kind: ImageStreamTag
          namespace: demo-project
          name: 'kube-docker-app:40904'
        lastTriggeredImage: >-
          adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c85505383
  replicas: 1
  revisionHistoryLimit: 10
  test: false
```

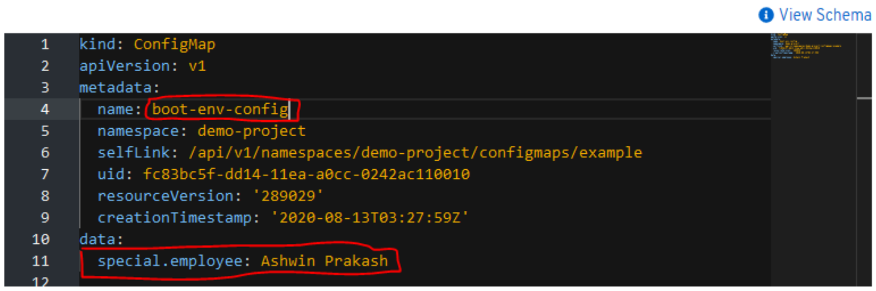
b. **Create Config Map as an Environment Variable**

Go to Workloads → Config Maps, Click on **Create Config Map**

In the Editor section shown below, add the red highlighted text starting the tab space shown at line no 7 and change the name at line no 4 as shown.

Click on **Create**



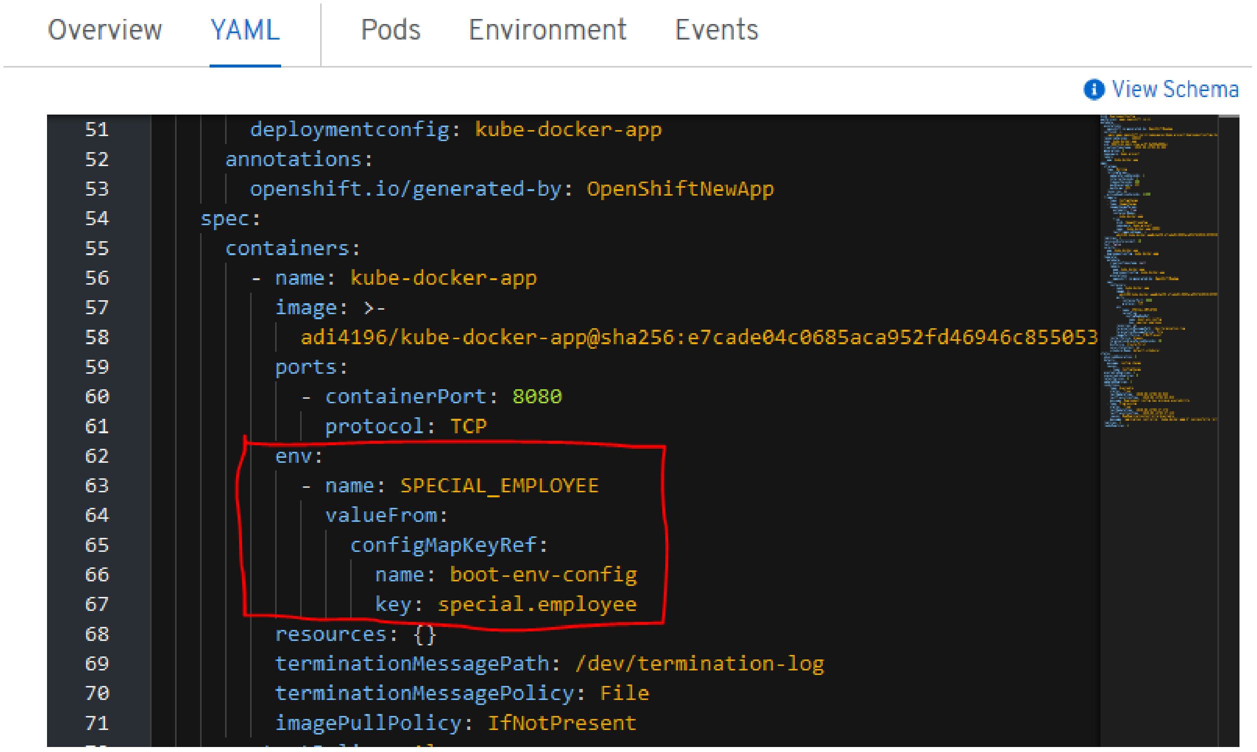


```
1   kind: ConfigMap
2   apiVersion: v1
3   metadata:
4     name: boot-env-config
5     namespace: demo-project
6     selfLink: /api/v1/namespaces/demo-project/configmaps/example
7     uid: fc83bc5f-dd14-11ea-a0cc-0242ac110010
8     resourceVersion: '289029'
9     creationTimestamp: '2020-08-13T03:27:59Z'
10  data:
11    special.employee: Ashwin Prakash
```

Go to Workloads → Deployment Configs → Name → YAML

Update the highlighted changes as shown below in the Deployment Config Editor

(Note:- Please maintain the indentation as shown in the below editor)



```
51          deploymentconfig: kube-docker-app
52        annotations:
53          openshift.io/generated-by: OpenShiftNewApp
54      spec:
55        containers:
56          - name: kube-docker-app
57            image: >-
58              adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c855053
59            ports:
60              - containerPort: 8080
61                protocol: TCP
62            env:
63              - name: SPECIAL_EMPLOYEE
64                valueFrom:
65                  configMapKeyRef:
66                    name: boot-env-config
67                    key: special.employee
68            resources: {}
69            terminationMessagePath: /dev/termination-log
70            terminationMessagePolicy: File
71            imagePullPolicy: IfNotPresent
```

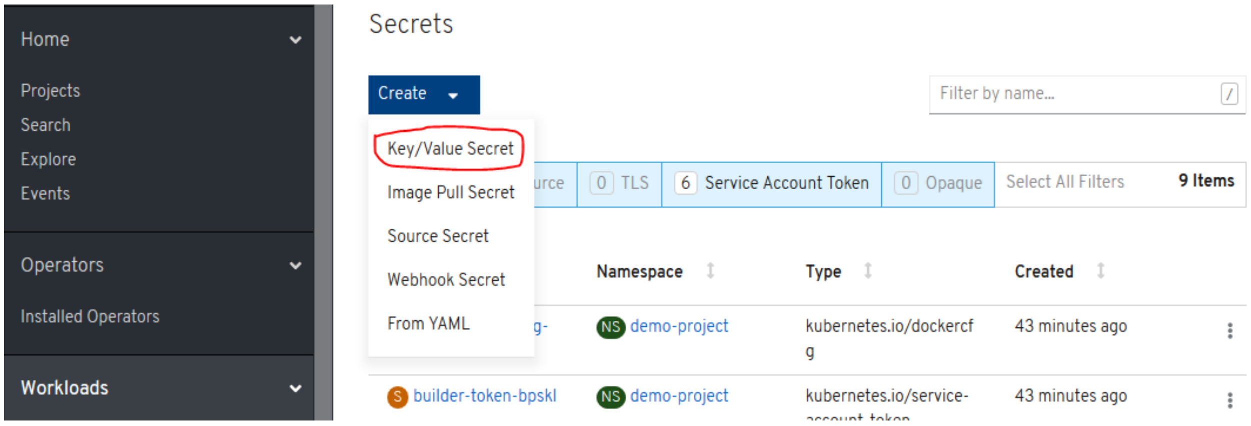
Use the same HostName which we created in the Route section and hit the below Url :-

'HostName/specialEmp'

c. Create Secret as an Environment Variable

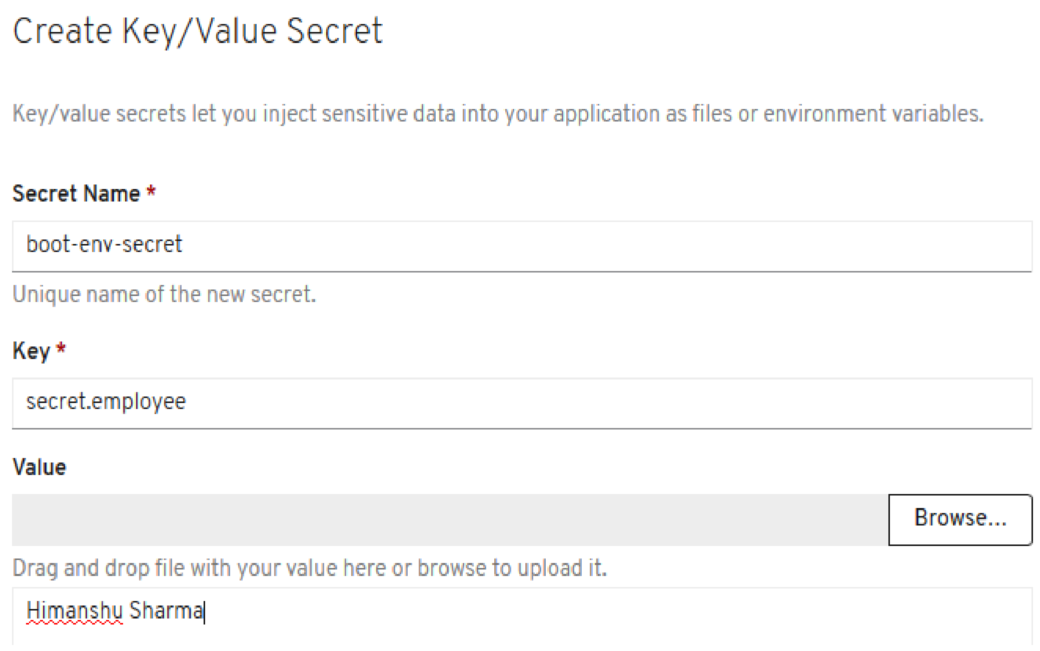
Go to Workloads → Secrets,

Select Key/Value Pair as the type as shown below.



	Namespace	Type	Created
g-	NS demo-project	kubernetes.io/dockercfg	43 minutes ago
S builder-token-bpskl	NS demo-project	kubernetes.io/service-account-token	43 minutes ago

Put the details as shown below and then click on Create



Create Key/Value Secret

Key/value secrets let you inject sensitive data into your application as files or environment variables.

Secret Name *

boot-env-secret

Unique name of the new secret.

Key *

secret.employee

Value

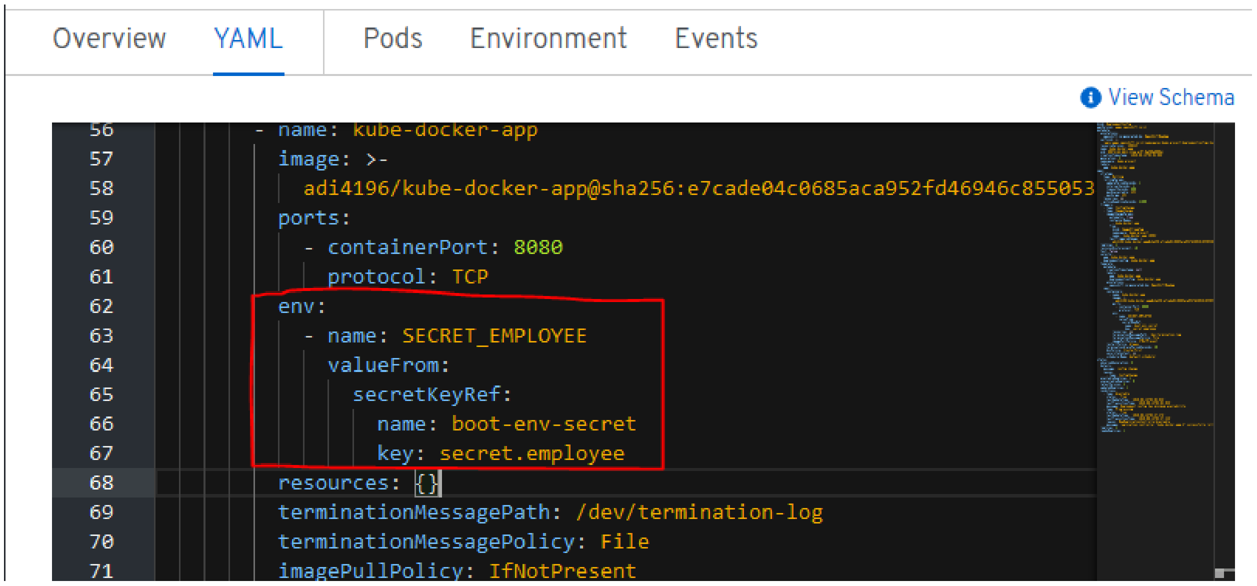
Drag and drop file with your value here or browse to upload it.

Himanshu Sharma

Go to Workloads → Deployment Configs → Name → YAML

Update the highlighted changes as shown below in the Deployment Config Editor

(Note:- Please maintain the indentation as shown in the below editor)



```
        - name: kube-docker-app
          image: >-
            adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c855053
          ports:
            - containerPort: 8080
              protocol: TCP
          env:
            - name: SECRET_EMPLOYEE
              valueFrom:
                secretKeyRef:
                  name: boot-env-secret
                  key: secret.employee
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          imagePullPolicy: IfNotPresent
```

Use the same HostName which we created in the Route section and hit the below Url :-

'HostName/secretEmp'