

# LAB UI MATERIAL

## DOCKER AND OPENSIFT

---

### 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/ashwin25/kube-docker-demo.git>
- c. `cd kube-docker-demo`  
`docker build --tag docker-service:[tagname] eg 1.0`  
(To
- d. **To push the built docker image to DockerHub (public docker image repository):-**  
`docker login -u ashwinprakash`  
`docker tag docker-service:1.0 ashwinprakash25/docker-service:1.0`  
`docker push ashwinprakash/docker-service:1.0`

Due to docker limits Ashwin is using quay.io

`docker login quay.io`

Username: ashwinprakash25

Password:

`docker tag docker-service:1.0 quay.io/ashwinprakash25/docker-service:1.0`

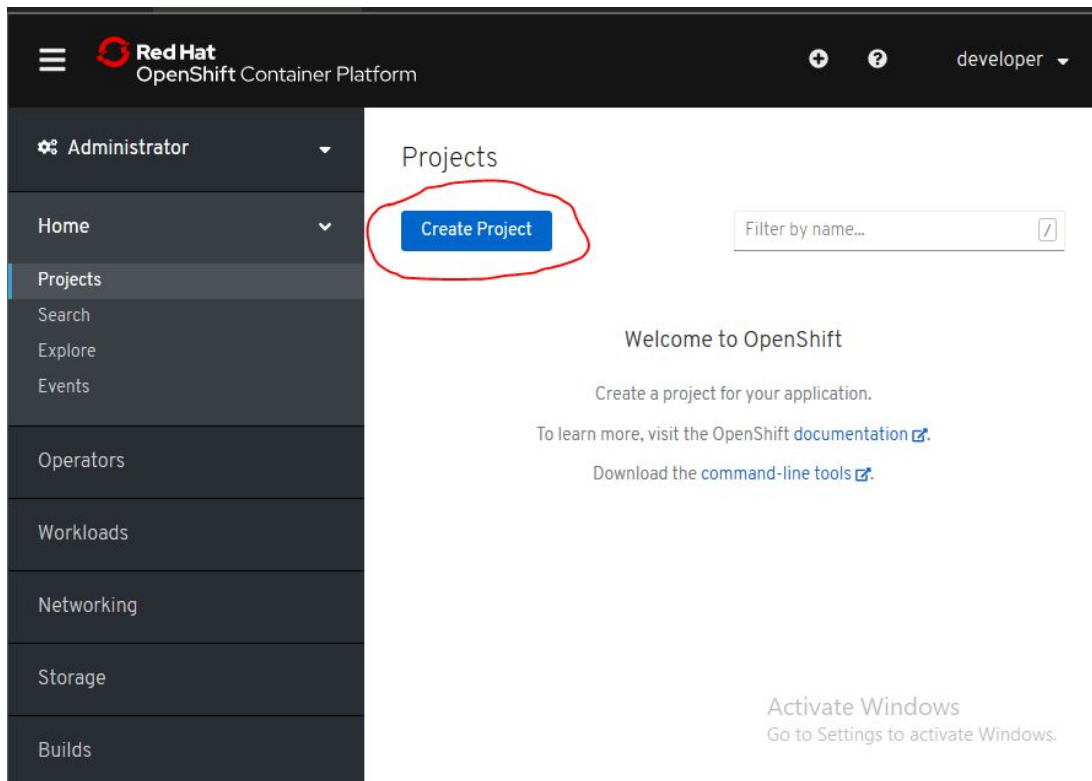
`docker push quay.io/ashwinprakash25/docker-service:1.0`

### 2. OPENSIFT 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**



- 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

- Explore
- Events
- Operators
- Workloads
- Pods**
- Deployments
- Deployment Configs
- Stateful Sets
- Secrets

1 Running
0 Pending
0 Terminating
0 CrashLoopBackOff
1 Completed
0 Failed

0 Unknown
[Select All Filters](#)

Name ↑	Namespace ↑	Pod Labels ↑	Node ↑
<span style="color: green;">P</span> kube-docker-app-1-mgvdv	<span style="color: green;">NS</span> 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

OverviewYAMLEnvironmentLogsEventsTerminal

Streaming events...

Showing 5 events

P kube-docker-app-1-mgdvdNS demo-project

Generated from kubelet on crc-rk2fc-master-0

Successfully pulled image "adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c8550538370d2be5745dc24a11dba6bd1d8c9f"

P kube-docker-app-1-mgdvdNS demo-project

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

The screenshot shows the Kubernetes dashboard interface. The 'Logs' tab is selected and highlighted with a red circle. Below the tabs, the 'Log streaming...' button is active, indicated by a green circle around the pause icon. The dropdown menu shows 'kube-docker-app'. To the right, there are 'Download' and 'Expand' buttons. The log output area shows 20 lines of logs, with the first line indicating a successful deployment of the 'nginx' container.

- 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

Home ▾

Projects

Search

Explore

Events

Operators

Workloads

Networking ▾

Services

**Routes**

Ingresses

Network Policies

Storage

## Create Route [Edit YAML](#)

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)

[Activate](#)  
Go to Settings

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

[Overview](#) [YAML](#)

## Route Overview

**Name**  
kube-docker-route

**Namespace**  
 demo-project

**Labels**  
app=kube-docker-app

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

**Status**  
 Accepted

### 3. OPENSIFT 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

Deployment Configs

Create Deployment Config

Filter by name...

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.

Overview **YAML** Pods Environment Events

View Schema

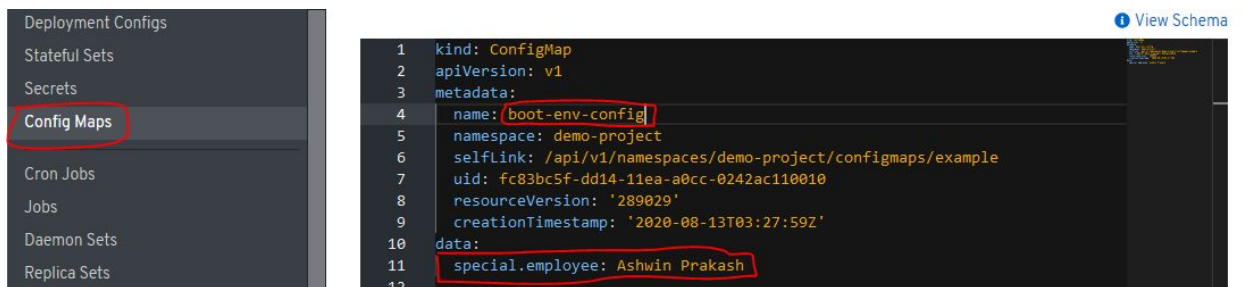
```
25 resources: {}
26 activeDeadlineSeconds: 21600
27 triggers:
28   - type: ConfigChange
29   - type: ImageChange
30     imageChangeParams:
31       automatic: true
32       containerNames:
33         - kube-docker-app
34       from:
35         kind: ImageStreamTag
36         namespace: demo-project
37         name: 'kube-docker-app:40904'
38       lastTriggeredImage: >-
39         adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c85505383
40     replicas: 1
41     revisionHistoryLimit: 10
42     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**

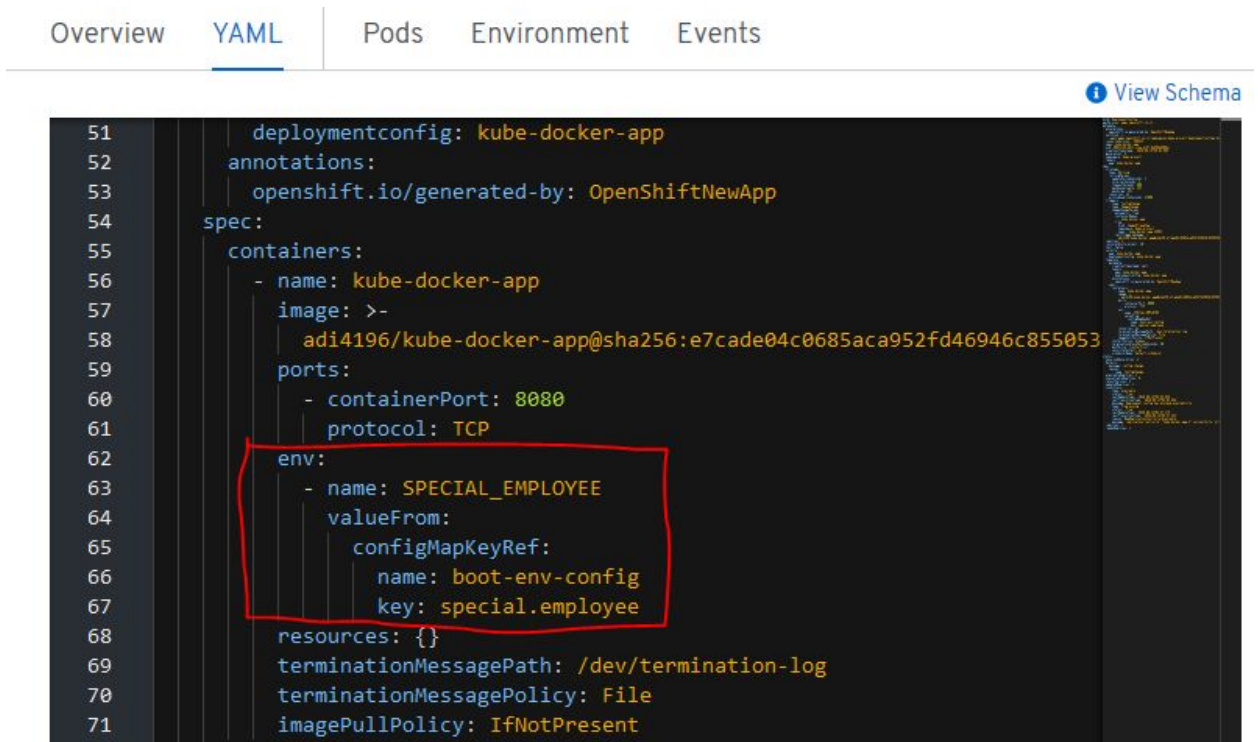


```
1 kind: ConfigMap
2 apiVersion: v1
3 metadata:
4   name: boot-env-config
5   namespace: demo-project
6   selflink: /api/v1/namespaces/demo-project/configmaps/example
7   uid: fc83bc5f-dd14-11ea-a0cc-0242ac110010
8   resourceVersion: '289029'
9   creationTimestamp: '2020-08-13T03:27:59Z'
10 data:
11   special.employee: Ashwin Prakash
12
```

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)



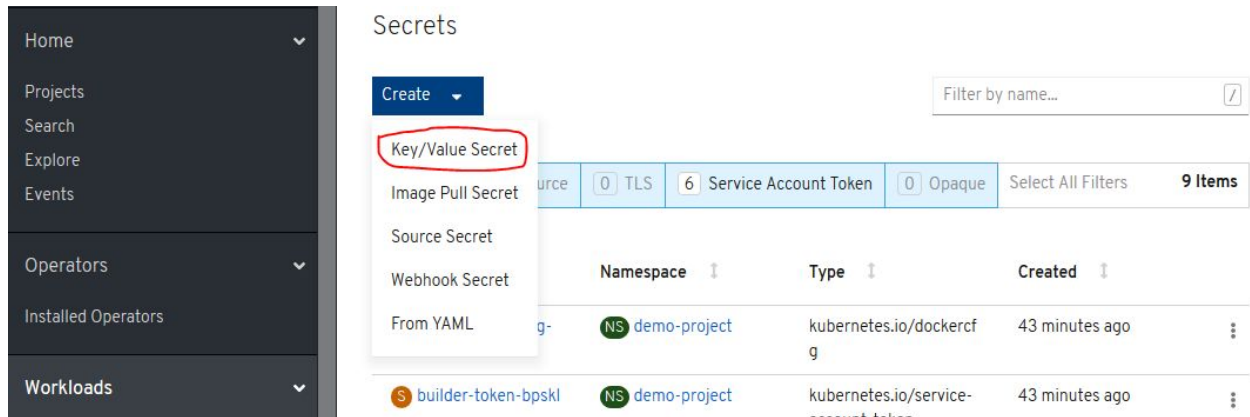
```
51 deploymentconfig: kube-docker-app
52 annotations:
53   openshift.io/generated-by: OpenShiftNewApp
54 spec:
55   containers:
56   - name: kube-docker-app
57     image: >-
58       adi4196/kube-docker-app@sha256:e7cade04c0685aca952fd46946c855053
59     ports:
60     - containerPort: 8080
61       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.



The screenshot shows the Kubernetes dashboard's 'Secrets' page. On the left is a sidebar with navigation links: Home, Projects, Search, Explore, Events, Operators, Installed Operators, and Workloads. The main content area is titled 'Secrets' and features a 'Create' button with a dropdown menu. The dropdown menu is open, showing options: 'Key/Value Secret' (highlighted with a red circle), 'Image Pull Secret', 'Source Secret', 'Webhook Secret', and 'From YAML'. To the right of the dropdown is a filter bar with 'Filter by name...' and a search icon. Below the filter bar is a table of secrets. The table has columns for 'Namespace', 'Type', and 'Created'. The first row shows a secret named 'demo-project' in the 'demo-project' namespace, of type 'kubernetes.io/dockerconfigjson', created 43 minutes ago. The second row shows a secret named 'builder-token-bpskl' in the 'demo-project' namespace, of type 'kubernetes.io/service-account-token', created 43 minutes ago.

Namespace	Type	Created
demo-project	kubernetes.io/dockerconfigjson	43 minutes ago
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 \*

Unique name of the new secret.

### Key \*

### Value

Browse...

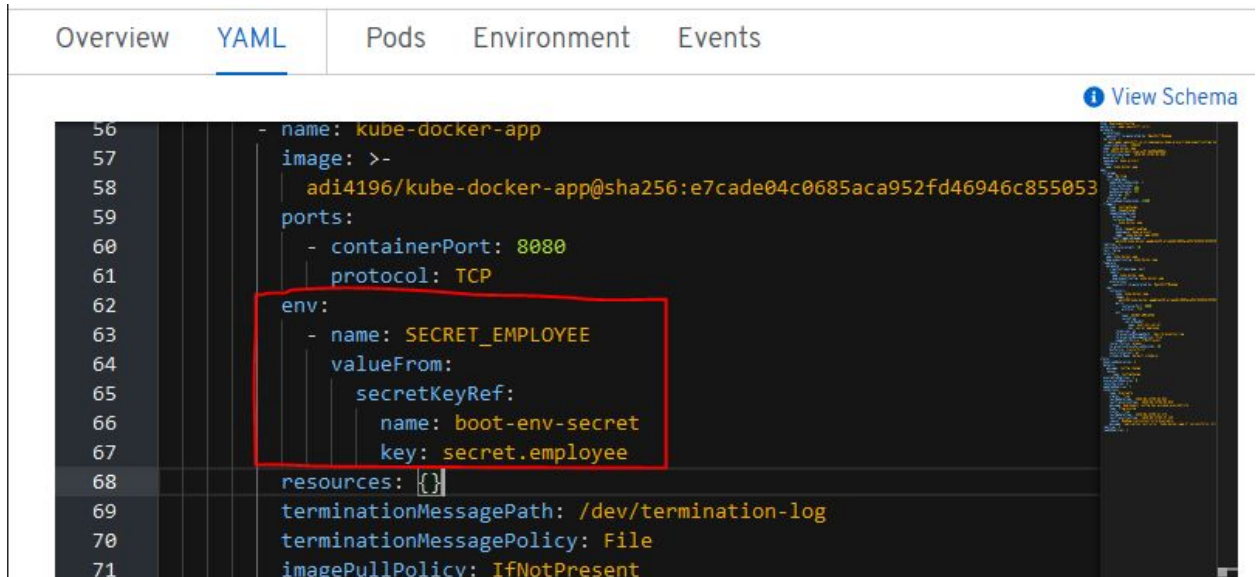
Drag and drop file with your value here or browse to upload it.

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)



The screenshot shows the 'YAML' tab in the Deployment Config Editor. The configuration is for a deployment named 'kube-docker-app'. The 'env' section is highlighted with a red box, showing the following configuration:

```
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: SECRET_EMPLOYEE
64       valueFrom:
65         secretKeyRef:
66           name: boot-env-secret
67           key: secret.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/secretEmp'