

# Optional: Deploy guestbook app from the OpenShift internal registry



## Objectives

In this lab, you will:

- Build and deploy a simple guestbook application
- Use OpenShift image streams to roll out an update
- Deploy a multi-tier version of the guestbook application

## Pre-requisite

You must have built and pushed the Guestbook application using the Docker commands as given in the Final Assignment.

## Deploy guestbook app from the OpenShift internal registry

As discussed in the course, IBM Cloud Container Registry scans images for common vulnerabilities and exposures to ensure that images are secure. But OpenShift also provides an internal registry – recall the discussion of image streams and image stream tags. Using the internal registry has benefits too. For example, there is less latency when pulling images for deployments. What if we could use both—use IBM Cloud Container Registry to scan our images and then automatically import those images to the internal registry for lower latency?

Please continue with the below commands from the steps where you left off in the previous lab.

1. Create an image stream that points to your image in IBM Cloud Container Registry.

1. 1

```
1. oc tag us.icr.io/$MY_NAMESPACE/guestbook:v1 guestbook:v1 --reference-policy=local --scheduled
```

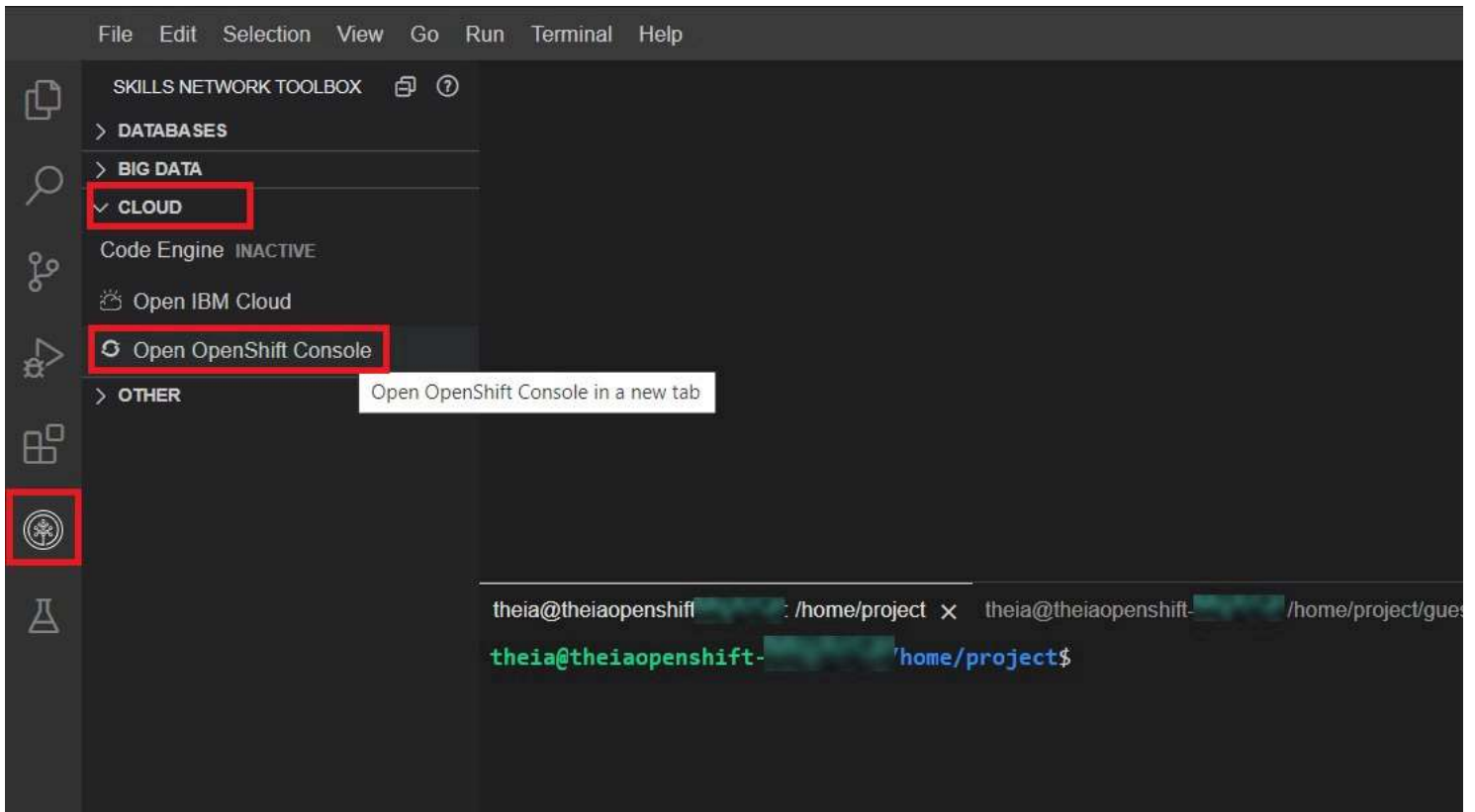
Copied!

With the `--reference-policy=local` option, a copy of the image from IBM Cloud Container Registry is imported into the local cache of the internal registry and made available to the cluster's projects as an image stream. The `--schedule` option sets up periodic importing of the image from IBM Cloud Container Registry into the internal registry. The default frequency is 15 minutes.

```
theia@theiaopenshift- [REDACTED]:/home/project/guestbook/v1/guestbook$ oc tag us.icr.io/$MY_NAMESPACE/guestbook:v1 guestbook:v1 --reference-policy=local --schedule=15m
Tag guestbook:v1 set to import us.icr.io/sn-labs-[REDACTED] guestbook:v1 periodically.
theia@theiaopenshift- [REDACTED]:/home/project/guestbook/v1/guestbook$
```

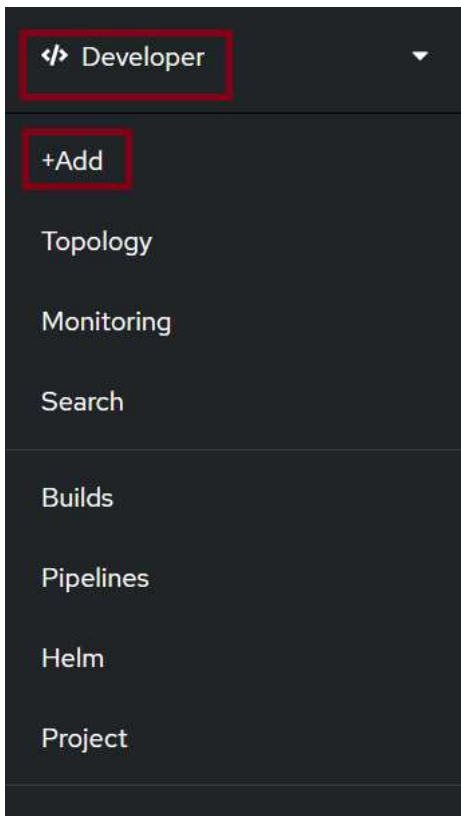
Now let's head over to the OpenShift web console to deploy the guestbook app using this image stream.

2. Click on the Skills Network Tool and select Cloud which will open a drop-down. Click on Open OpenShift console which will open the Open Shift Web console in a new window.



**Note:** Currently we are experiencing certain difficulties with the OpenShift console . If your screen takes time in loading, please close the OpenShift console browser tab & re-launch the same. It may take upto 10 mins to load the screen.

3. From the Developer perspective, click the **+Add** button to add a new application to this project.



4. Click the **Container Image** option so that we can deploy the application using an image in the internal registry.

## Add

Select a way to create an application, component or service from one of the options.

The screenshot shows the 'Add' page with several deployment options. The 'Container Image' option is highlighted with a red box. The options are:

- Quick Starts**: Deploying an application with a pipeline, Getting started with a sample, Adding health checks to your sample application. [See all Quick Starts](#)
- Samples**: Create an application from a code sample
- From Git**: Import code from your Git repository to be built and deployed
- Container Image**: Deploy an existing image from an image registry or image stream tag
- From Catalog**: Browse the catalog to discover, deploy and connect to services
- Database**: Browse the catalog to discover database services to add to your application
- Operator Backed**: Browse the catalog to discover and deploy operator managed services
- Helm Chart**: Browse the catalog to discover and install Helm Charts
- Pipeline**: Create automated application

5. Under **Image**, switch to “Image stream tag from internal registry”.

## Deploy Image

### Image

Deploy an existing image from an image stream or image registry.

☒ Image name from external registry

Enter an image name

To deploy an image from a private repository, you must [create an image pull secret](#) with your image registry credentials.

☐ Allow images from insecure registries

☐ Image stream tag from internal registry

6. Select your project, and the image stream and tag you just created (guestbook and v1, respectively). You should have only have one option for each of these fields anyway since you only have access to a single project and you only created one image stream and one image stream tag.

### Image


Deploy an existing image from an image stream or image registry.

☐ Image name from external registry

☒ Image stream tag from internal registry

Project *	Image Stream *	Tag *
sn-labs-	guestbook	v1

Runtime Icon

 openshift

The icon represents your image in Topology view. A label will also be added to the resource defining the icon.

7. Keep all the default values and hit **Create** at the bottom. This will create the application and take you to the Topology view.

## Image

Deploy an existing image from an image stream or image registry.

- ☐ Image name from external registry
- ☒ Image stream tag from internal registry

Project *	Image Stream *	Tag *
sn-labs-	guestbook	v1

### Runtime Icon

 openshift

The icon represents your image in Topology view. A label will also be added to the resource defining the icon.

## General

### Application Name

guestbook-app

A unique name given to the application grouping to label your resources.

### Name \*

guestbook

A unique name given to the component that will be used to name associated resources.

## Resources

Select the resource type to generate

- ☒ Deployment

apps/Deployment

A Deployment enables declarative updates for Pods and ReplicaSets.

8. From the Topology view, click the guestbook Deployment. This should take you to the **Resources** tab for this Deployment, where you can see the Pod that is running the application as well as the Service and Route that expose it.

**Kindly wait as the deployments in the Topology view may take time to get running.**

The screenshot displays the OpenShift console interface for the 'guestbook' resource. On the left, a topography view shows two resource icons: 'guestbook' (a blue circle with a red 'C') and 'opnsh...onsole' (a blue circle with a red 'C'). The 'guestbook' icon is highlighted with a red box. On the right, the 'guestbook' resource details are shown. At the top, there's a 'Health Checks' warning: 'Container guestbook does not have health checks to ensure your application is running correctly. Add Health Checks'. Below this, there are tabs for 'Details', 'Resources', and 'Monitoring'. The 'Resources' tab is selected, showing sections for 'Pods', 'Builds', 'Services', and 'Routes'. The 'Pods' section shows one pod: 'guestbook-56dbf59579-mz2l5' in a 'Running' state. The 'Builds' section shows 'No Build Configs found for this resource.' The 'Services' section shows a service 'guestbook' with 'Service port: 3000-tcp → Pod Port: 3000'. The 'Routes' section shows a route 'RT guestbook' with a location URL: 'http://guestbook-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud'.

**Note:** Kindly do not delete the `opnsh.console` deployment in the Topography view as this is essential for the OpenShift console to function properly.

9. Click the Route location (the link) to view the guestbook in action.

**Note:** Please wait for status of the pod to change to **'Running'** before launching the app.

The screenshot displays the OpenShift console interface for a deployment named 'guestbook'. On the left, a visual overview shows the 'guestbook' deployment (blue circle with a red 'C') and its associated 'guestbook-app' (green circle with a red 'A'). To the right, the 'Resources' tab is active, showing details for the 'guestbook' deployment. A 'Health Checks' notification indicates that the container does not have health checks. Below this, the 'Pods' section shows a single pod 'guestbook-56dbf59579-mz215' in a 'Running' state. The 'Builds' section shows no build configs. The 'Services' section shows a service 'guestbook' with port 3000. The 'Routes' section, highlighted with a red border, shows a route 'RT guestbook' with the location 'http://guestbook-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud'.

**guestbook** Actions

**Health Checks** ×  
Container guestbook does not have health checks to ensure your application is running correctly. [Add Health Checks](#)

Details **Resources** Monitoring

**Pods**

**guestbook-56dbf59579-mz215** Running [View logs](#)

**Builds**  
No Build Configs found for this resource.

**Services**

**guestbook**  
Service port: 3000-tcp → Pod Port: 3000

**Routes**

**RT guestbook**  
Location:  
<http://guestbook-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud>

10. Try out the guestbook by putting in a few entries. You should see them appear above the input box after you hit **Submit**.

# Guestbook - v1

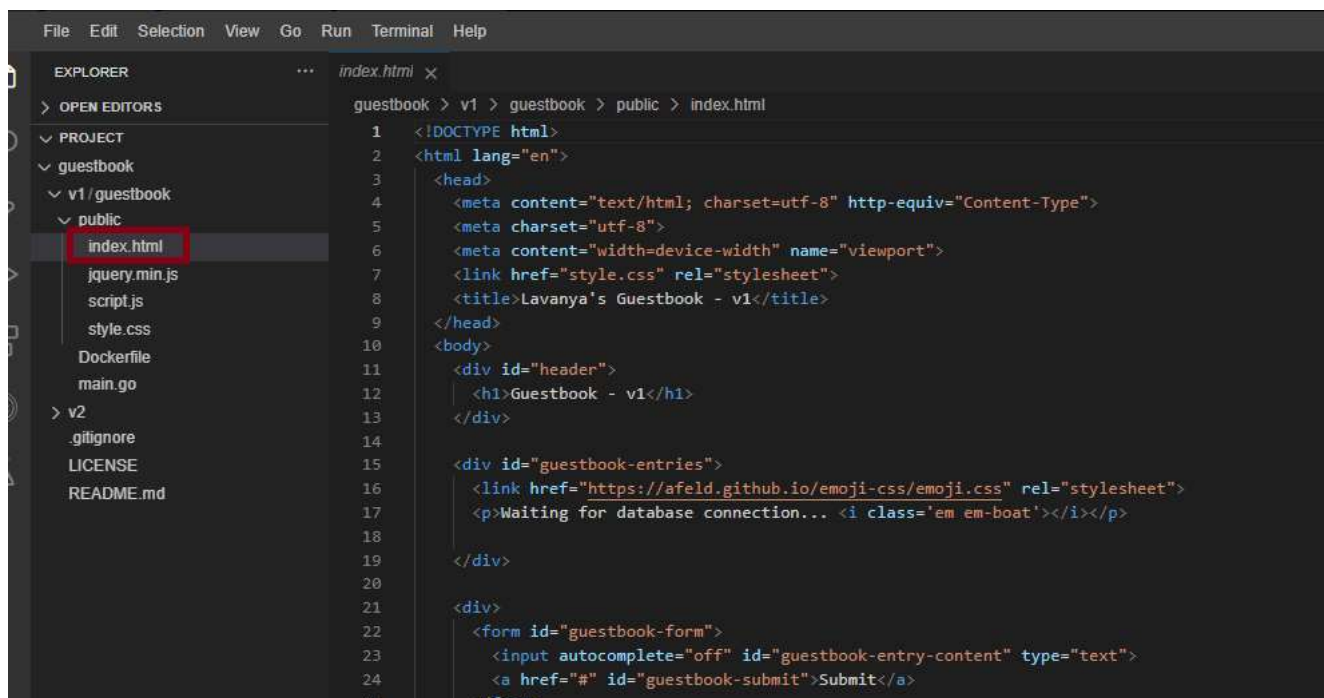
hello  
wishes for the day!

http://guestbook-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud  
[/env](#) [/info](#)

## Update the guestbook

Let's update the guestbook and see how OpenShift's image streams can help us update our apps with ease.

1. Use the Explorer to edit `index.html` in the `public` directory. The path to this file is `guestbook/v1/guestbook/public/index.html`.



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The file structure is as follows:

- PROJECT
  - guestbook
    - v1/guestbook
      - public
        - index.html** (highlighted with a red box)
        - jquery.min.js
        - script.js
        - style.css
- v2
  - .gitignore
- LICENSE
- README.md

The main editor displays the content of `index.html`:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
5     <meta charset="utf-8">
6     <meta content="width=device-width" name="viewport">
7     <link href="style.css" rel="stylesheet">
8     <title>Lavanya's Guestbook - v1</title>
9   </head>
10  <body>
11    <div id="header">
12      <h1>Guestbook - v1</h1>
13    </div>
14
15    <div id="guestbook-entries">
16      <link href="https://afeld.github.io/emoji-css/emoji.css" rel="stylesheet">
17      <p>Waiting for database connection... <i class='em em-boat'></i></p>
18    </div>
19
20
21    <div>
22      <form id="guestbook-form">
23        <input autocomplete="off" id="guestbook-entry-content" type="text">
24        <a href="#" id="guestbook-submit">Submit</a>
25      </form>
26    </div>
27  </body>
28 </html>
```



- Let's edit the title to be more specific. On line number 12, that says `<h1>Guestbook - v1</h1>`, change it to include your name. Something like `<h1>Alex's Guestbook - v1</h1>`. Make sure to save the file when you're done.

```

index.html x
guestbook > v1 > guestbook > public > index.html
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
5      <meta charset="utf-8">
6      <meta content="width=device-width" name="viewport">
7      <link href="style.css" rel="stylesheet">
8      <title>Lavanya's Guestbook - v1</title>
9    </head>
10   <body>
11     <div id="header">
12       <h1>Alex's Guestbook - v1</h1>
13     </div>
14
15     <div id="guestbook-entries">
16       <link href="https://sfeld.github.io/emoji-css/emoji.css" rel="stylesheet">
17       <p>Waiting for database connection... <i class='em em-boat'></i></p>
18     </div>
19
20     <div>
21       <form id="guestbook-form">
22         <input autocomplete="off" id="guestbook-entry-content" type="text">
23         <a href="#" id="guestbook-submit">Submit</a>
24       </form>
25     </div>
26
27   </body>
28 </html>

```

- Build and push the app again using the same tag. This will overwrite the previous image.

- 1

1. `docker build . -t us.icr.io/$MY_NAMESPACE/guestbook:v1 && docker push us.icr.io/$MY_NAMESPACE/guestbook:v1`

Copied!

```

theia@theiaopenshift: /home/project/guestbook/v1/guestbook$ docker build . -t us.icr.io/$MY_NAMESPACE/guestbook:v1 && docker
ook:v1
Sending build context to Docker daemon  98.3kB
Step 1/14 : FROM golang:1.15 as builder
--> 40349a2425ef
Step 2/14 : RUN go get github.com/codegangsta/negroni
--> Using cache
--> a7657fc96c64
Step 3/14 : RUN go get github.com/gorilla/mux github.com/xyproto/simpleredis
--> Using cache
--> 1f28b8fef54e
Step 4/14 : COPY main.go .
--> Using cache
--> 3458048c5c1e
Step 5/14 : RUN go build main.go
--> Using cache
--> 823973fe49e6
Step 6/14 : FROM ubuntu:18.04
--> f5cbcd4244ba
Step 7/14 : COPY --from=builder /go//main /app/guestbook
--> Using cache
--> 0350722f466e
Step 8/14 : ADD public/index.html /app/public/index.html
--> a9644611258d
Step 9/14 : ADD public/script.js /app/public/script.js
--> 89215eb5fecb
Step 10/14 : ADD public/style.css /app/public/style.css
--> 7760ade3c7d3
Step 11/14 : ADD public/jquery.min.js /app/public/jquery.min.js
--> 97abdf76f88d
Step 12/14 : WORKDIR /app
--> Running in 1b78236b819b
Removing intermediate container 1b78236b819b
--> 6f056cbe5b7

```

- Recall the `--schedule` option we specified when we imported our image into the OpenShift internal registry. As a result, OpenShift will regularly import new images pushed to the specified tag. Since we pushed our newly built image to the same tag, OpenShift will import the updated image within about 15 minutes. If you don't want to wait for OpenShift to automatically import the image, run the following command.

- 1

1. `oc import-image guestbook:v1 --from=us.icr.io/$MY_NAMESPACE/guestbook:v1 --confirm`

Copied!



```

theia@theiaopenshift- /home/project/guestbook/v1/guestbook$ oc import-image guestbook:v1 --from=us.icr.io/$MY_NAMESPACE/guest
imagestream.image.openshift.io/guestbook imported

Name:                guestbook
Namespace:            sn-labs-
Created:              2 minutes ago
Labels:               <none>
Annotations:          openshift.io/image.dockerRepositoryCheck=2022-04-11T08:28:50Z
Image Repository:     image-registry.openshift-image-registry.svc:5000/sn-labs- guestbook
Image Lookup:         local=false
Unique Images:        2
Tags:                 1

v1
updates automatically from registry us.icr.io/sn-labs- /guestbook:v1
prefer registry pullthrough when referencing this tag

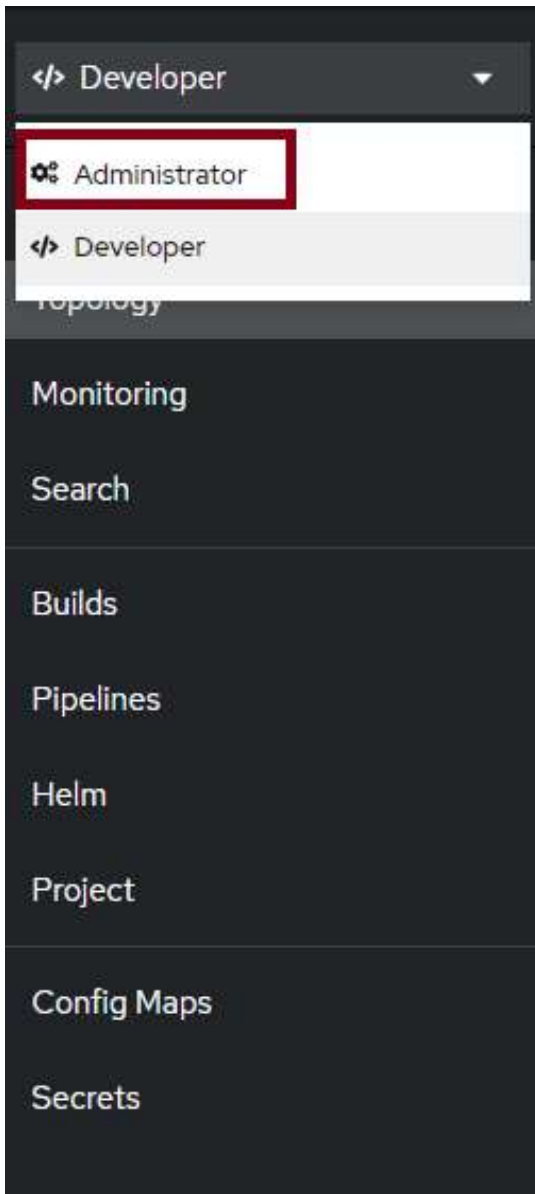
* us.icr.io/sn-labs- /guestbook@sha256:f16401f8452ae414e5feafbb621ef20a0b9a12aafc800679465d809525ee454d
  Less than a second ago
  us.icr.io/sn-labs- /guestbook@sha256:11ee56c6d46d80f0fda0790f50121bf3a4760240d74baef01a5c6bded6e94ef7
  2 minutes ago

Image Name:          guestbook:v1
Docker Image:        us.icr.io/sn-labs- /guestbook@sha256:f16401f8452ae414e5feafbb621ef20a0b9a12aafc800679465d809525ee454d
Name:                sha256:f16401f8452ae414e5feafbb621ef20a0b9a12aafc800679465d809525ee454d
Created:              Less than a second ago
Annotations:          image.openshift.io/dockerLayersOrder=ascending
Image Size:           31.53MB in 6 layers
Layers:              26.71MB sha256:08a6abff89437fab99b52abbefed82ea907f12845c30eeb94f6b93c69be93166
                     4.791MB sha256:1158f7aa125a353046e120906d415a3f0e7b2fb43fa61ffed49645f9eb423948
                     650B   sha256:98e5b3b04689c7d80aebec108d7d88619d06fe3771382c6fa44d7273b74d5faa
                     608B   sha256:e3277739aac4aaac2fbabf0e282914400e62c5e67da5cea7825eb5b0a519aca0
                     545B   sha256:5a480e70e8fe40c001754b4143e4bb13a945b38e5adf7082a83ef70145d45b0d
                     29.88kB sha256:5e702cd921b48b45d5f1ad2fa8d75390a12189f4c249acf995cc5185f3015a83

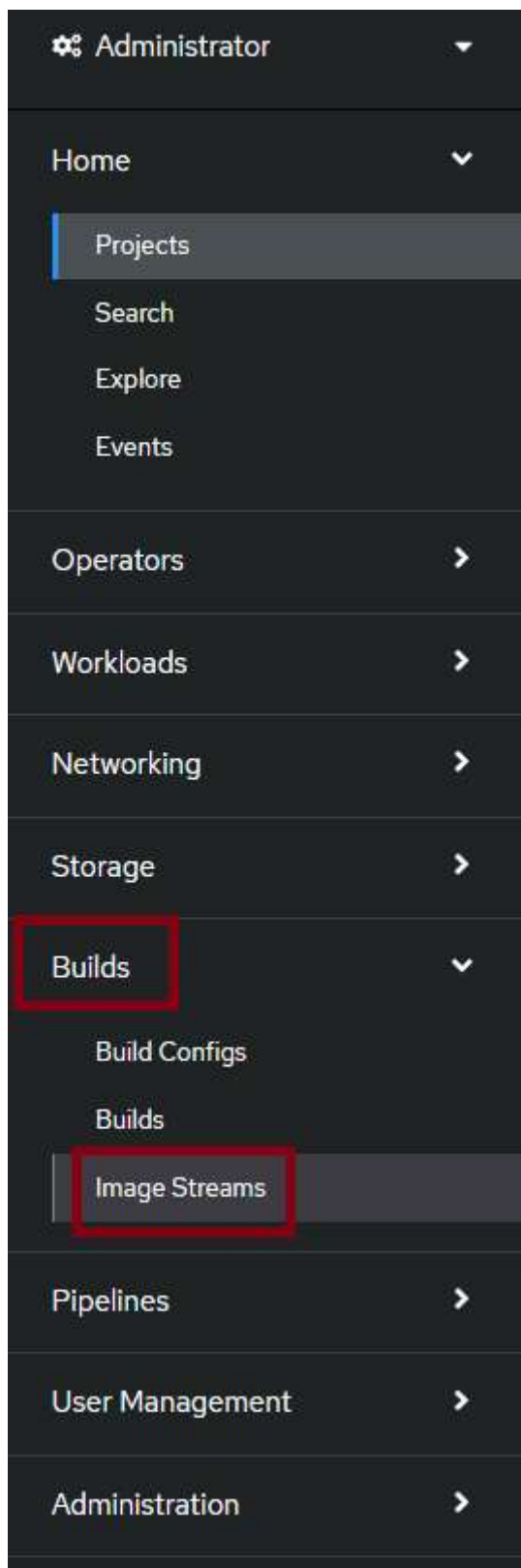
Image Created:        28 seconds ago
Author:                <none>
Arch:                 amd64

```

5. Switch to the **Administrator** perspective so that you can view image streams.



6. Click **Builds** > **Image Streams** in the navigation.



7. Click the guestbook image stream.

## Image Streams

Name

Search by name...

/

Name	Labels	Created
 guestbook	No labels	 Apr 1



8. Click the **History** menu. If you only see one entry listed here, it means OpenShift hasn't imported your new image yet. Wait a few minutes and refresh the page. Eventually you should see a second entry, indicating that a new version of this image stream tag has been imported. This can take some time as the default frequency for importing is 15 minutes.



Image Streams > Image Stream Details

 guestbook

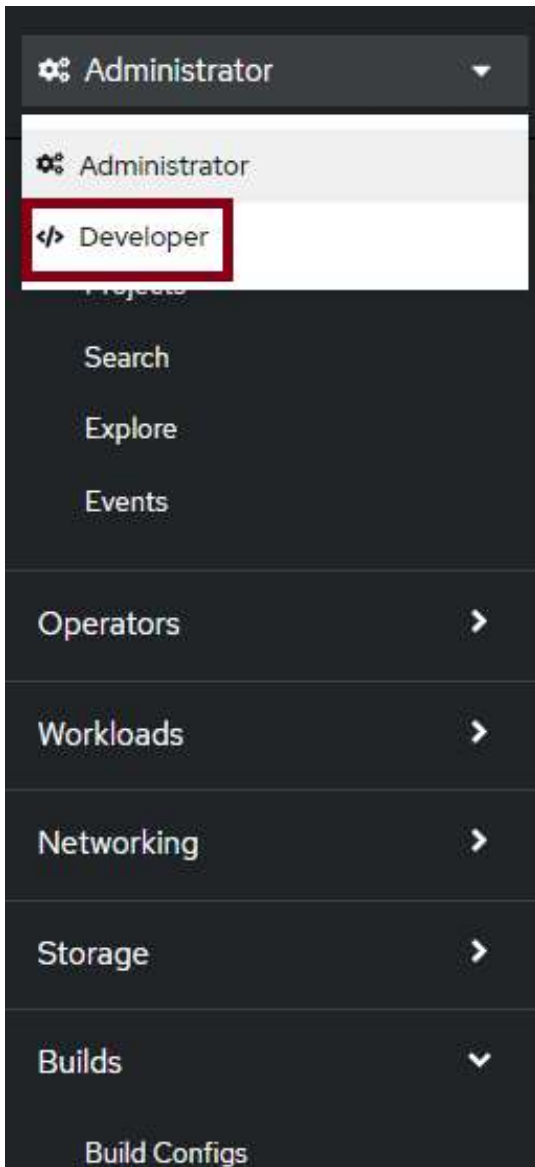
Details **History**

- a minute ago

 guestbook:v1  
from us.icr.io/sn-labs-/guestbook  
sha256:f16401f8452ae414e5feafbb621ef20a0b9a12aafc800679465d809525ee454d
- 3 minutes ago

 guestbook:v1  
from us.icr.io/sn-labs-/guestbook  
sha256:11ee56c6d46d80f0fda0790f50121bf3a4760240d74baef01a5c6bde6e94ef7

9. Return to the **Developer** perspective.



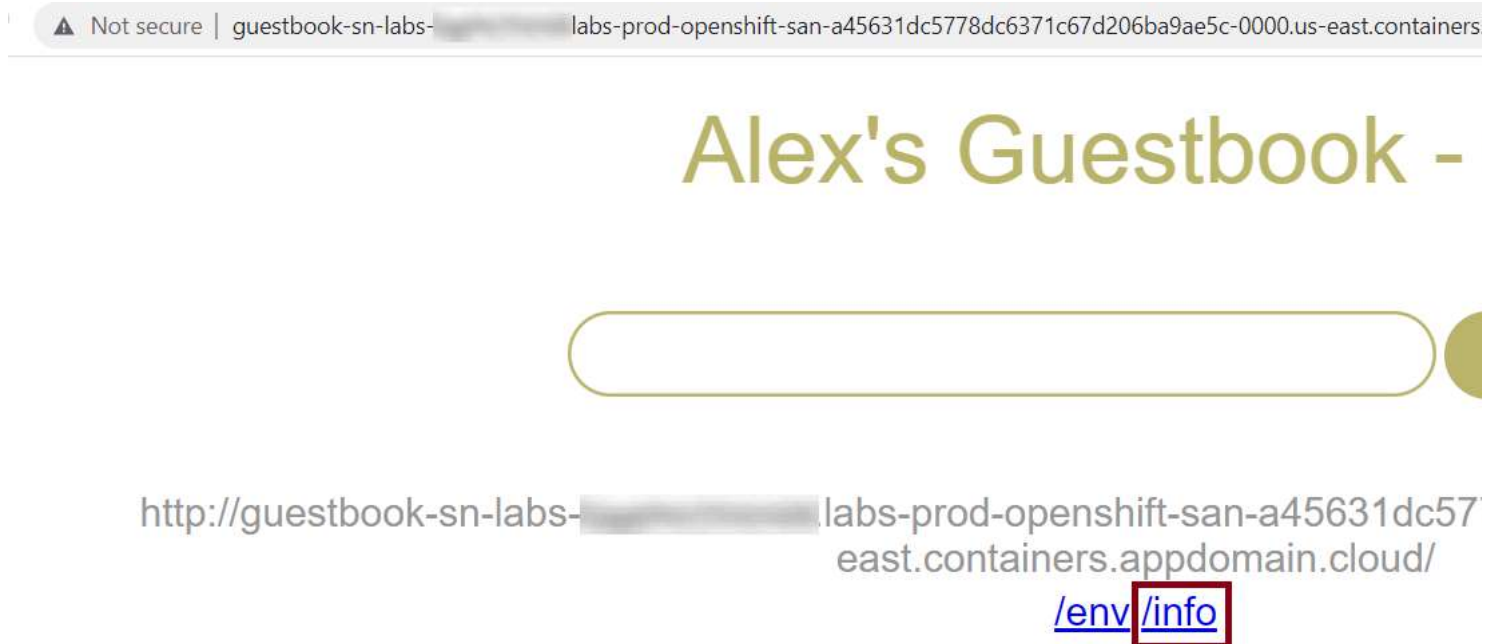
**Note: Please wait for some time for the OpenShift console & the Developer perspective to load.**

10. View the guestbook in the browser again. If you still have the tab open, go there. If not, click the Route again from the guestbook Deployment. You should see your new title on this page! OpenShift imported the new version of our image, and since the Deployment points to the image stream, it began running this new version as well.

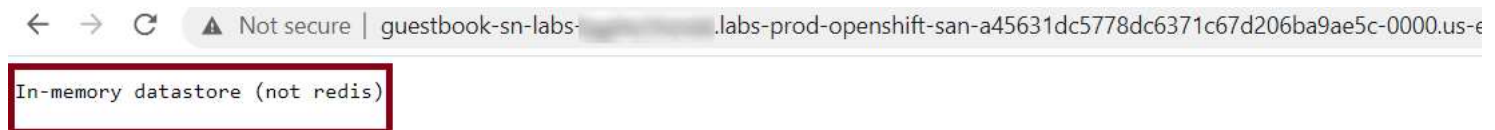


## Guestbook storage

1. From the guestbook in the browser, click the `/info` link beneath the input box. This is an information endpoint for the guestbook.



Notice that it says “In-memory datastore (not redis)”. Currently, we have only deployed the guestbook web front end, so it is using in-memory datastore to keep track of the entries. This is not very resilient, however, because any update or even a restart of the Pod will cause the entries to be lost. But let’s confirm this.



2. Return to the guestbook application in the browser by clicking the Route location again. You should see that your previous entries appear no more. This is because the guestbook was restarted when your update was deployed in the last section. We need a way to persist the guestbook entries even after restarts.

# Alex's Guestbook



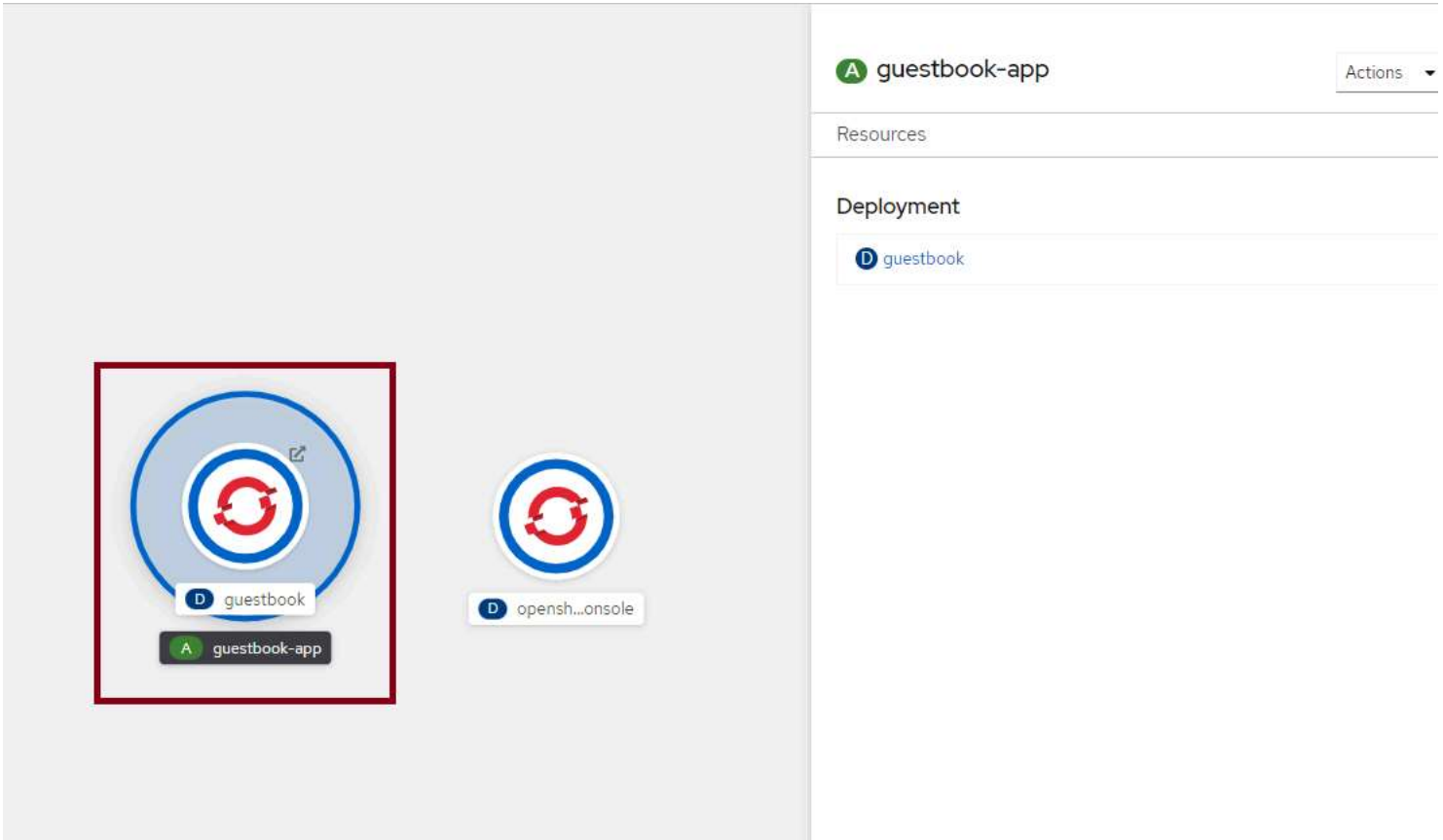
http://guestbook-sn-labs-...labs-prod-openshift-san-a45631dc!  
east.containers.appdomain.cloud/

**Note:** Currently we are experiencing certain difficulties with the OpenShift console. There is a possibility that you will see your old entries because the image stream takes time in updating. You may move ahead with the further steps of lab.

## Delete the guestbook

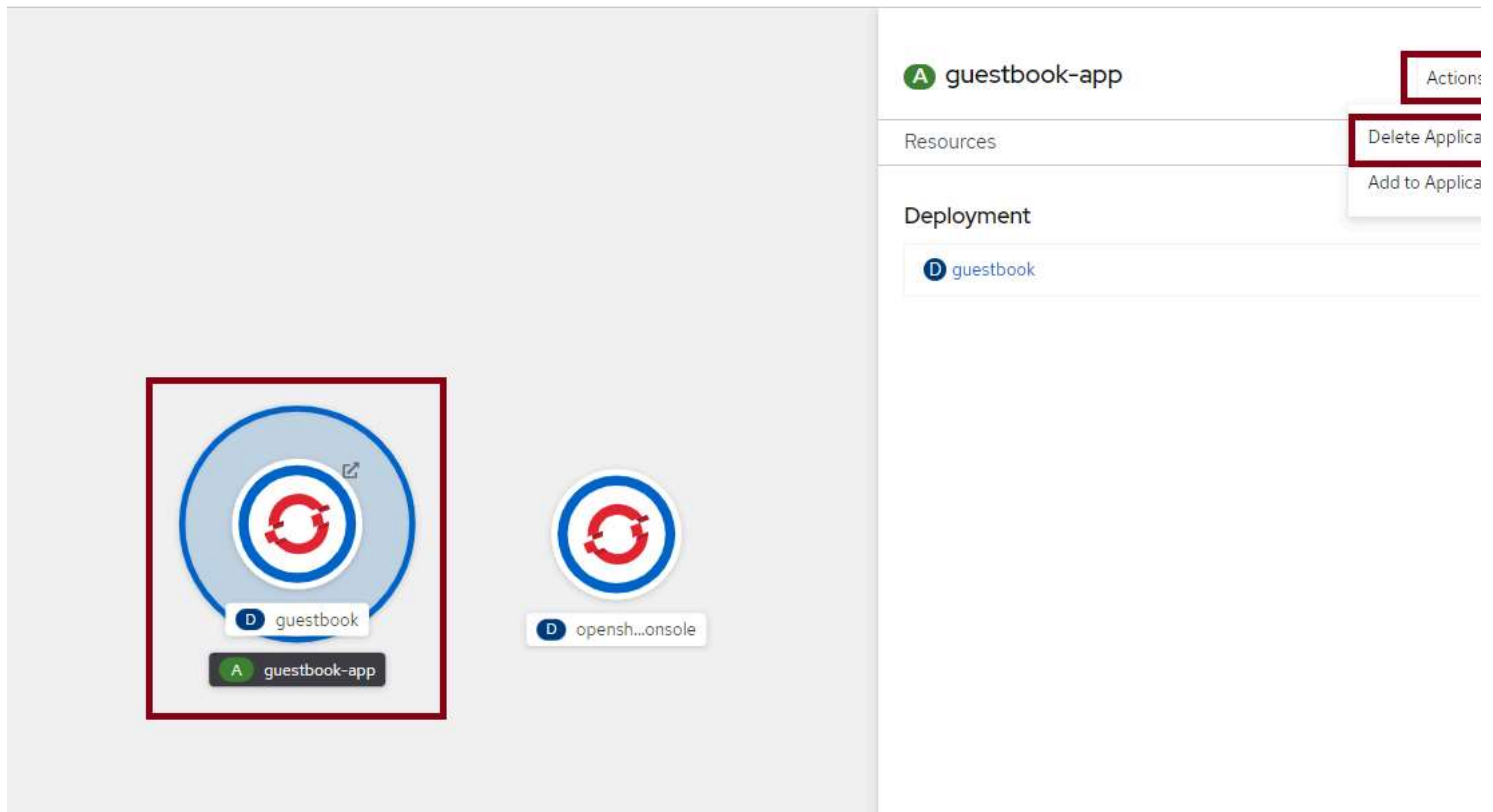
In order to deploy a more complex version of the guestbook, delete this simple version.

1. From the Topology view, click the `guestbook-app` application. This is the light gray circle that surrounds the `guestbook` Deployment.

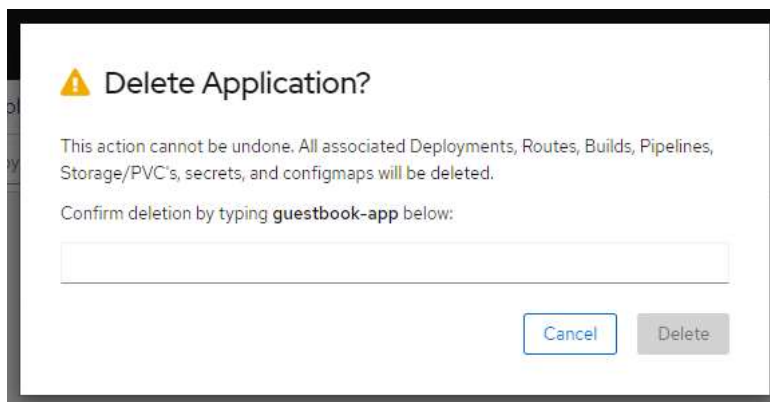


2. Click **Actions > Delete Application**.





3. Type in the application name and click **Delete**.



## Deploy Redis master and slave

We've demonstrated that we need persistent storage in order for the guestbook to be effective. Let's deploy Redis so that we get just that. Redis is an open source, in-memory data structure store, used as a database, cache and message broker.

This application uses the v2 version of the guestbook web front end and adds on 1) a Redis master for storage and 2) a replicated set of Redis slaves. For all of these components, there are Kubernetes Deployments, Pods, and Services. One of the main concerns with building a multi-tier application on Kubernetes is resolving dependencies between all of these separately deployed components.

In a multi-tier application, there are two primary ways that service dependencies can be resolved. The v2/guestbook/main.go code provides examples of each. For Redis, the master endpoint is discovered through environment variables. These environment variables are set when the Redis services are started, so the service resources need to be created before the guestbook Pods start. Consequently, we'll follow a specific order when creating the application components. First, the Redis components will be created, then the guestbook application.

**Note:** If you have tried this lab earlier, there might be a possibility that the previous session is still persistent. In such a case, you will see an **'Unchanged'** message instead of the **'Created'** message when you run the **Apply** command for creating deployments. We recommend you to proceed with the next steps of the lab.

1. From the terminal in the lab environment, change to the v2 directory.

- ```
1. 1
1. cd ../../v2
```

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v1/guestbook$ cd ../../v2
theia@theiaopenshift- /home/project/guestbook/v2$
```

2. Run the following command or open the redis-master-deployment.yaml in the Explorer to familiarize yourself with the Deployment configuration for the Redis master.

1. 1

1. cat redis-master-deployment.yaml

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ cat redis-master-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
      role: master
  template:
    metadata:
      labels:
        app: redis
        role: master
    spec:
      containers:
        - name: redis-master
          image: redis:5.0.5
          ports:
            - name: redis-server
              containerPort: 6379
          volumeMounts:
            - name: redis-storage
              mountPath: /data
      volumes:
        - name: redis-storage
          emptyDir: {}
theia@theiaopenshift- /home/project/guestbook/v2$
```

3. Create the Redis master Deployment.

1. 1

1. oc apply -f redis-master-deployment.yaml

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ oc apply -f redis-master-deployment.yaml
deployment.apps/redis-master created
theia@theiaopenshift- /home/project/guestbook/v2$
```

4. Verify that the Deployment was created.

1. 1

1. oc get deployments

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ oc get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
openshift-web-console 1/1     1             1          14m
redis-master        1/1     1             1           34s
theia@theiaopenshift- /home/project/guestbook/v2$
```

5. List Pods to see the Pod created by the Deployment.

1. 1

1. oc get pods

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
openshift-web-console-77d78f965-6kr12 2/2     Running   0           14m
redis-master-d98597c5b-f2g54           1/1     Running   0           58s
theia@theiaopenshift- /home/project/guestbook/v2$
```

You can also return to the Topology view in the OpenShift web console and see that the Deployment has appeared there.

6. Run the following command or open the `redis-master-service.yaml` in the Explorer to familiarize yourself with the Service configuration for the Redis master.

1. 1

1. `cat redis-master-service.yaml`

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ cat redis-master-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
spec:
  ports:
  - port: 6379
    targetPort: redis-server
  selector:
    app: redis
    role: master
theia@theiaopenshift- /home/project/guestbook/v2$
```

Services find the Pods to load balance based on Pod labels. The Pod that you created in previous step has the labels `app=redis` and `role=master`. The selector field of the Service determines which Pods will receive the traffic sent to the Service.

7. Create the Redis master Service.

1. 1

1. `oc apply -f redis-master-service.yaml`

Copied!

```
theia@theiaopenshift- /home/project/guestbook/v2$ oc apply -f redis-master-service.yaml
service/redis-master created
theia@theiaopenshift- /home/project/guestbook/v2$
```

If you click on the `redis-master` Deployment in the Topology view, you should now see the `redis-master` Service in the **Resources** tab.

The screenshot shows the OpenShift web console interface. On the left, the Topology view displays a cluster of resources. A red box highlights the 'redis-master' Deployment icon. On the right, the details panel for the 'redis-master' resource is shown. The 'Resources' tab is selected, and a red box highlights the 'Services' section, which lists the 'redis-master' Service. The service configuration shows 'Service port: TCP/6379' and 'Pod Port: redis-server'. A 'Health Checks' warning is visible at the top of the details panel, stating 'Container redis-master does not have health checks to ensure your application is running correctly. Add Health Checks'.

8. Run the following command or open the `redis-slave-deployment.yaml` in the Explorer to familiarize yourself with the Deployment configuration for the Redis slave.

1. 1

1. `cat redis-slave-deployment.yaml`

Copied!

```

theia@theiaopenshift: /home/project/guestbook/v2$ cat redis-slave-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-slave
  labels:
    app: redis
    role: slave
spec:
  replicas: 2
  selector:
    matchLabels:
      app: redis
      role: slave
  template:
    metadata:
      labels:
        app: redis
        role: slave
    spec:
      containers:
        - name: redis-slave
          image: redis:5.0.5
          command: ["/bin/sh"]
          args: ["-c", "redis-server --slaveof redis-master 6379"]
          ports:
            - name: redis-server
              containerPort: 6379
          volumeMounts:
            - name: redis-storage
              mountPath: /data
      volumes:
        - name: redis-storage
          emptyDir: {}
theia@theiaopenshift: /home/project/guestbook/v2$

```

9. Create the Redis slave Deployment.

1. 1

1. oc apply -f redis-slave-deployment.yaml

Copied!

```

theia@theiaopenshift: /home/project/guestbook/v2$ oc apply -f redis-slave-deployment.yaml
deployment.apps/redis-slave created
theia@theiaopenshift: /home/project/guestbook/v2$

```

10. Verify that the Deployment was created.

1. 1

1. oc get deployments

Copied!

```

theia@theiaopenshift: /home/project/guestbook/v2$ oc get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
openshift-web-console 1/1     1            1           25m
redis-master         1/1     1            1           12m
redis-slave          2/2     2            2           29s
theia@theiaopenshift: /home/project/guestbook/v2$

```

11. List Pods to see the Pod created by the Deployment.

1. 1

1. oc get pods

Copied!

```

theia@theiaopenshift: /home/project/guestbook/v2$ oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
openshift-web-console-77d78f965-6kr12 2/2     Running   0          26m
redis-master-d98597c5b-f2g54          1/1     Running   0          13m
redis-slave-76dfcf6864-7qkmg          1/1     Running   0          66s
redis-slave-76dfcf6864-lqnnq          1/1     Running   0          66s
theia@theiaopenshift: /home/project/guestbook/v2$

```

You can also return to the Topology view in the OpenShift web console and see that the Deployment has appeared there.

12. Run the following command or open the redis-slave-service.yaml in the Explorer to familiarize yourself with the Service configuration for the Redis slave.

1. 1

1. cat redis-slave-service.yaml

Copied!

```
theia@theiaopenshift: /home/project/guestbook/v2$ cat redis-slave-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    app: redis
    role: slave
spec:
  ports:
    - port: 6379
      targetPort: redis-server
  selector:
    app: redis
    role: slave
```

13. Create the Redis slave Service.

1. 1

1. `oc apply -f redis-slave-service.yaml`

Copied!

```
theia@theiaopenshift: /home/project/guestbook/v2$ oc apply -f redis-slave-service.yaml
service/redis-slave created
```

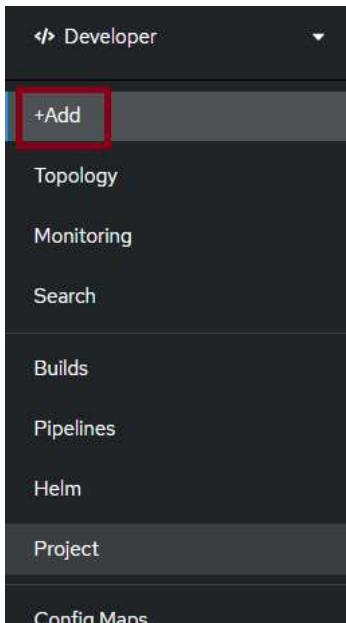
If you click on the redis-slave Deployment in the Topology view, you should now see the redis-slave Service in the **Resources** tab.

The screenshot shows the OpenShift console interface. On the left, the 'Topology' view displays three deployment icons: 'redis-master', 'opensh...onsole', and 'redis-slave'. The 'redis-slave' icon is highlighted with a red rectangular box. On the right, the 'Details' panel for the 'redis-slave' deployment is shown. It includes a 'Health Checks' section with a warning that the container does not have health checks. Below this, the 'Resources' tab is active, showing a table of 'Pods' with two entries: 'redis-slave-76dfcf6864-7qkmg' and 'redis-slave-76dfcf6864-lqnnq', both in a 'Running' state. Below the pods, the 'Services' section is highlighted with a red rectangular box, showing a single service entry: 'redis-slave' with the configuration 'Service port: TCP/6379 → Pod Port: redis-server'. The 'Routes' section below it shows 'No Routes found for this resource.'

## Deploy v2 guestbook app

Now it's time to deploy the second version of the guestbook app, which will leverage Redis for persistent storage.

1. Click the **+Add** button to add a new application to this project.

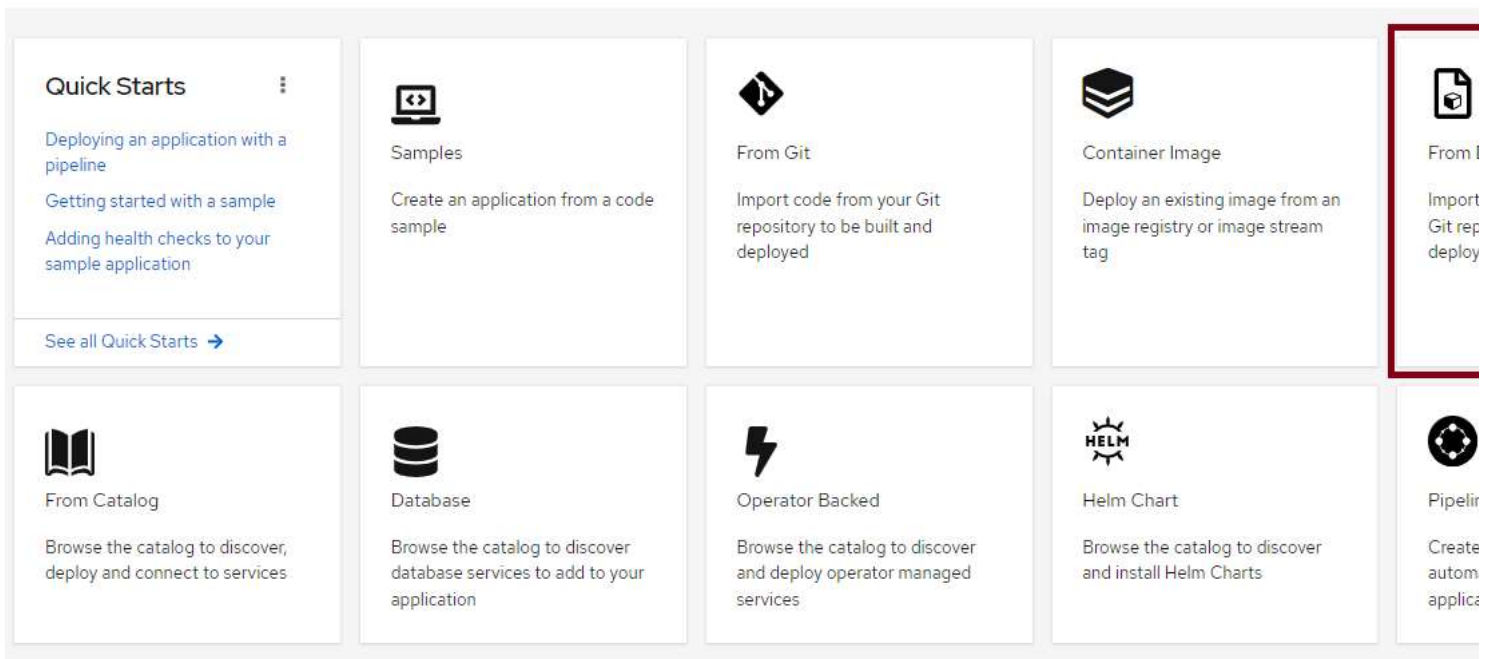


To demonstrate the various options available in OpenShift, we'll deploy this guestbook app using an OpenShift build and the Dockerfile from the repo.

2. Click the **From Dockerfile** option.

## Add

Select a way to create an application, component or service from one of the options.



3. Paste the below URL in the **Git Repo URL** box.

1. 1

1. <https://github.com/ibm-developer-skills-network/guestbook>

Copied!

You should see a validated checkmark once you click out of the box.

**Note: Ensure there are no spaces in the Git Repo URL that is to be copied.**



## Import from Dockerfile

### Git

Git Repo URL \*

https://github.com/ibm-developer-skills-network/guestbook.git



Validated

› [Show Advanced Git Options](#)

4. Click **Show Advanced Git Options**.

## Import from Dockerfile

### Git

Git Repo URL \*

https://github.com/ibm-developer-skills-network/guestbook.git



Validated

› [Show Advanced Git Options](#)

5. Since the Dockerfile isn't at the root of the repository, we need to tell OpenShift where it is. Enter /v2/guestbook in the **Context Dir** box.

▼ [Hide Advanced Git Options](#)

Git Reference

Optional branch, tag, or commit.

Context Dir

/v2/guestbook

Optional subdirectory for the application source code, used as a context directory for build.

6. Under **Container Port**, enter 3000.

Optional subdirectory for the application source code, used as a context directory for build.

Source Secret

Select Secret Name



Secret with credentials for pulling your source code.

### Dockerfile

Dockerfile Path

Dockerfile

Allows the builds to use a different path to locate your Dockerfile, relative to the Context Dir field.

Container Port


3000

Port number the container exposes.

7. Leave the rest of the default options and click **Create**.


Since we gave OpenShift a Dockerfile, it will create a BuildConfig and a Build that will build an image using the Dockerfile, push it to the internal registry, and use that image for a Deployment.




Project: sn-labs 

---

### Git

**Git Repo URL \***  
  
Validated 

 Hide Advanced Git Options

**Git Reference**  
  
Optional branch, tag, or commit.

**Context Dir**  
  
Optional subdirectory for the application source code, used as a context directory for build.

**Source Secret**  
  
Secret with credentials for pulling your source code.

### Dockerfile

**Dockerfile Path**  
  
Allows the builds to use a different path to locate your Dockerfile, relative to the Context Dir field.

**Container Port**  
  
Port number the container exposes.

8. From the Topology view, click the guestbook Deployment.

In the **Resources** tab, click the Route location to load the guestbook in the browser. Notice that the header says “Guestbook - v2” instead of “Guestbook - v1”.

**Note: Please wait for the Builds to complete before clicking on the route link**

**guestbook-git**

**Health Checks**  
Container guestbook-git does not have health checks to ensure application is running correctly. [Add Health Checks](#)

Details Resources Monitoring

**Pods**

**Waiting for the build**  
Waiting for the first build to run successfully. You may temporarily see "ImagePullBackOff" and "ErrImagePull" errors while waiting.  
[Show waiting pods with errors](#)

No Pods found for this resource.

**Builds**

**guestbook-git**

Build #1 is complete (a few seconds ago)

**Services**

**guestbook-git**  
Service port: 3000-tcp → Pod Port: 3000

**Routes**

**guestbook-git**  
Location:  
<http://guestbook-git-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud/>

9. From the guestbook in the browser, click the `/info` link beneath the input box.

⚠ Not secure | guestbook-git-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.contain

# Guestbook - v2

<http://guestbook-git-sn-labs-...labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud/>  
[/env](#) [/info](#)

Notice that it now gives information on Redis since we're no longer using the in-memory datastore.



```
# Server
redis_version:5.0.5
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:442b43d467cd2b03
redis_mode:standalone
os:Linux 3.10.0-1160.59.1.el7.x86_64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:8.3.0
process_id:1
run_id:ddfb01eefa82ace731a16c65dae1ddc6cbe031d3
tcp_port:6379
uptime_in_seconds:1637
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:5500602
executable:/data/redis-server
config_file:

# Clients
connected_clients:1
client_recent_max_input_buffer:2
client_recent_max_output_buffer:0
blocked_clients:0

# Memory
used_memory:1944584
used_memory_human:1.85M
used_memory_rss:10461184
used_memory_rss_human:9.98M
used_memory_peak:1963600
used_memory_peak_human:1.87M
used_memory_peak_perc:99.03%
used_memory_overhead:1923354
used_memory_startup:791240
used_memory_dataset:21230
used_memory_dataset_perc:1.84%
allocator_allocated:1919168
allocator_active:2138112
allocator_resident:5660672
```

- If you wish to delete & redeploy your app on Openshift due to session persistence or other errors, please follow the steps given [here](#)

After doing the above, if you do not see any route to your guestbook app, please run the below command in the terminal to get the app route:

1. 1
1. oc status

Copied!

The guestbook app may show a 'Waiting for Database connection' status for some time after clicking on the route, due to which, the entries added in the box will not appear. You may have to wait for sometime for the app to be ready and then add your entries to see them appear correctly.

© IBM Corporation. All rights reserved.