Deploying Microservices

Estimated time: 90 minutes

Welcome to the lab **Deploying Microservices**. You will deploy the **Pictures** and **Songs** services to the cloud in this lab. You should have all the code ready from the previous labs in this capstone. The lab focuses on deploying working code as follows:

- **Pictures** service is deployed to IBM Code Engine.
- **Songs** service is deployed to RedHat OpenShift.

Learning Objectives:

After completing this lab you will be able to:

1. Start Code Engine service in the lab environment.
2. Deploy a Flask service to Code Engine.
3. Access the RedHat OpenShift platform in the lab environment.
4. Deploy a Flask service to RedHat OpenShift.

Exercise 1: Pictures - Start Code Engine

Your Tasks

1. On the menu in your lab environment, click the `Cloud` dropdown menu and select `Code Engine`. The code engine setup panel appears. Click `Create Project` to begin.

2. The code engine environment takes a while to prepare. You will see the progress status is indicated in the setup panel.

3. Once the code engine set up is complete, you can see that it is active. Click `Code Engine CLI` to begin the pre-configured CLI in the terminal as shown below.

   Take a screenshot of the terminal and save it as `deploy-getpic-1.jpg`.

4. You will observe that the pre-configured CLI startup and the home directory are set to the current directory. As a part of the pre-configuration, the project has been set up, and Kubeconfig is set up. The details are shown on the terminal as follows.

Evidence

1. Take a screenshot of the terminal showing the output of the `ibmcloud ce project current` command.
2. Save the screenshot as `deploy-getpic-1.jpg` (or .png).

Exercise 2: Pictures - Clone GitHub Repository

The next step is to clone the **Pictures** microservice repository to the local lab environment.

Your Tasks

You created a new repository for the pictures service from the provided template in a previous lab. If not, go back to the `Create Get Pictures Service with Flask` lab and ensure you complete it before coming back to this lab.

1. Open a terminal with `Terminal -> New Terminal` if one is not open already.

2. Next, use the export GITHUB_ACCOUNT command to export an environment variable that contains the name of your GitHub account.

   Note: Substitute your real GitHub account for the {your_github_account} placeholder below:

   ```
   export GITHUB_ACCOUNT={your_github_account}
   ```

3. Then use the following commands to clone your repository.

   ```
   git clone https://github.com/$GITHUB_ACCOUNT/Back-End-Development-Pictures.git
   ```

4. Click the button below and see if the `routes.py` contains all the changes you have done and pushed in the previous lab. If you don't see the changes, you have to repeat the previous lab and rebuild you deployment on the Openshift container.

Open **routes.py** in IDE

Exercise 3: Pictures - Run Application locally

Your Tasks

Now that you have all the code in the local lab environment, let's run the applicaiton locally to ensure it works as expected. You should have finished implementing the microservice in a previous lab.

Task 1: Run the application locally

1. Change to your project directory as follows.

   ```
   cd Back-End-Development-Pictures
   ```

2. Run locally to test once by running the following command.

```
flask run --debugger --reload
```

3. Open another new terminal and run the following command to check the application health.

```
curl localhost:5000/health
```

If the application does not run locally, you should return to module 1 and ensure you have finished the `Create Get Pictures Service with Flask` lab. Once you are done testing locally, you can exit the server by using the keys `ctrl+c` on the keyboard. You can exit the terminal by using the `exit` command.

Exercise 4: Pictures - Deploy to Code Engine

Once you have ensured that the application is running as expected, the next thing is to deploy the application on Code Engine. This will require the Dockerfile, which has been provided already. You will first build an image of the application and push it to the lab image registry provided under your account.

Your Tasks

1. The lab environment comes with an IBM Container Registry. You can push images to only one namespace ${SN_ICR_NAMESPACE}. Use the terminal to find what this namespace is for your lab environment:

```
echo ${SN_ICR_NAMESPACE}
```

You should see an output similar to this:

```
theia@theiaopenshift-captainfedo1:/home/project$ echo ${SN_ICR_NAMESPACE}
sn-labs-captainfedo1
```

2. Build the image with the following command:

```
docker build -t pictures .
```

3. Tag the image as us.icr.io/$SN_ICR_NAMESPACE/pictures:1.

```
docker tag pictures us.icr.io/$SN_ICR_NAMESPACE/pictures:1
```

4. You can see both the original and the tagged images by using:

```
docker images
```

Your output should have the two images with any additional images already provided by the lab environment:

```
$ docker images
```

```
REPOSITORY                                TAG          IMAGE ID       CREATED         SIZE
pictures                                  latest       7c230d5bcdbf   2 minutes ago   179MB
us.icr.io/sn-labs-captainfedo1/pictures   1            7c230d5bcdbf   2 minutes ago   179MB
```

Note that the orignal image and the tagged images have the same `IMAGE ID` indicating the difference is in tags only.

5. Push the image to the registry using the following command:

```
docker push us.icr.io/$SN_ICR_NAMESPACE/pictures:1
```

6. You can use the following command to list all images in your namespace.

```
ibmcloud cr images --restrict $SN_ICR_NAMESPACE
```

You should see the image you just pushed:

```
$ ibmcloud cr images --restrict $SN_ICR_NAMESPACE
Listing images...
Repository                                Tag   Digest         Namespace              Created         Size    Security status
us.icr.io/sn-labs-captainfedo1/pictures   1     f3c8e1ef47f4   sn-labs-captainfedo1   7 minutes ago   68 MB   -
OK
```

7. Use the following command to deploy the application on Code Engine:

```
ibmcloud ce app create --name pictures  --image us.icr.io/${SN_ICR_NAMESPACE}/pictures:1 --registry-secret icr-secret --port 3000
```

You will see that the command creates the application and also internally sets up the required infrastructure. It takes a few minutes and it finally gives a confirmation along with the URL. Your URL will look different:

```
$ ibmcloud ce app create --name pictures  --image us.icr.io/${SN_ICR_NAMESPACE}/pictures:1 --registry-secret icr-secret --port 3000
Creating application 'pictures'...
The Route is still working to reflect the latest desired specification.
Configuration 'pictures' is waiting for a Revision to become ready.
Ingress has not yet been reconciled.
Waiting for load balancer to be ready.
Run 'ibmcloud ce application get -n pictures' to check the application status.
OK
https://pictures.zcx38jqwqmu.us-south.codeengine.appdomain.cloud
```

Take a screenshow of the terminal showing the output of the `ibm ce app create --` command and save it is `deploy-getpic-2.png` (or jpg).

8. Copy the application URL, paste it in a new tab, and press **Enter**. You will need to add a valid path to see a result. You can test `URL/count` or `URL/health` to test the microservice. Take a screenshot of the browser tab showing the `/count` endpoint and save it as `deploy-getpic-3.png` (or .jpg).

9. If you lose the application URL, you can look it up by using the command:

```
ibmcloud ce application list
```

You should see the link in the URL column:

```
$ ibmcloud ce application list
Listing all applications...
OK
Name      Status  URL                                                      Latest          Age    Conditions  Reason
pictures  Ready   https://pictures.zcx38jqwqmu.us-south.codeengine.appdomain.cloud  pictures-00001  4m55s  3 OK / 3
```

Evidence

1. Take a screenshot of the terminal showing the output of the `ibmcloud ce create ...` command with the final URL for the application and save it as `deploy-getpic-2.png` (or .jpg).
2. Take a screenshot of the running application in your browser with the `/count` endpoint. Save the screenshot as `deploy-getpic-3.png` (or .jpg).
3. Save the URL of the pictures service. You will use it in the next lab to connect the main Django application with the microservice.

Exercise 5: Pictures - Confirm Code Engine Application

Congratulations on running your application on Code Engine. This final step will verify the application using the terminal.

Your Tasks

1. To get the details of the application, run the following command:

```
ibmcloud ce app get --name pictures
```

You should see detailed information about the application, including the time of creation, resources used, number of instances, and other details.

Exercise 6: Songs - Install MongoDB on OpenShift

You used the MongoDb server provided in the lab environment for the exercises in the previous modules. This server is limited to the lab environment only and cannot be reached from another platform like RedHat OpenShift. In order to successfully use MongoDb as the database server in production, you will install it in RedHat OpenShift in this exercise.

Your Tasks

1. Launch the OpenShift console from the lab environment. This will open a new tab.

If you see a start dialog, you can simply `Cancel` out of it.

2. Click `Topology`. You should see only one application in your project.
3. Choose from the left menu and switch to the **Administrator** option.

3. Navigate through the options in the left-hand menu to access **Builds**, then proceed to click and open the **imageStreams** section.

4. Select the option to generate an ImageStream by clicking the **Create ImangeStream** button button.

5. Copy and paste the provided code into the editor, then proceed to select the option to create.

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: mongo
spec:
  lookupPolicy:
    local: false
  tags:
    - name: latest
      from:
        kind: DockerImage
        name: docker.io/library/mongo:latest
```

6. Next, navigate to the **Developer** mode using the options available in the left-hand menu.

7. Select the **“Add+”** option by clicking on it.

8. Choose **“Container images”** to initiate the deployment of the image generated in the previous step.

9. Choose the **“Image stream tag from internal registry”** option, then within the imagestream, select **“mongo”** and for the tag, choose **“latest”** Maintain the remaining options unchanged and proceed by clicking on the **“create”** button.

Take a screenshot of the `Image` page showing the correct Container Image as `mongo` and Tag as `latest`. Save it as `deploy-getsong-1.png` (or .jpg)

10. You can observe the deployment of the MongoDB server on the OpenShift platform by navigating to the Topology page.

Take a screenshot of the topology view showing Mongo running as an application and save as `deploy-getsong-2.png` (or .jpg).

11. Go back to the lab tab and open the terminal in this lab if not already opened.

Evidence

1. Take a screenshot of the `Image` page showing the correct Container Image as `mongo` and Tag as `latest`. Save it as `deploy-getsong-1.png` (or .jpg)
2. Take a screenshot of the topology view showing Mongo running as an application. Save it as `deploy-getsong-2.png` (or .jpg).

Optional Tasks

1. We did not expose the `mongo` application to the internet. However, you can access this server from within the pod itself. Execute the following command in the terminal to print out the mongo version:

   ```
   oc get po
   ```

   You should see an output as follows:

   ```
   oc get po
   NAME                                       READY   STATUS    RESTARTS   AGE
   mongo-79cbc7d7b9-5x5p9                     1/1     Running   0          18m
   openshift-web-console-78f49566d7-4bs8m     2/2     Running   0          26m
   openshift-web-console-78f49566d7-68dkp     2/2     Running   0          26m
   ```

2. Run the following command with the mongo pod. The name assigned to your pod will be different than what is shown here:

   ```
   oc exec mongo-79cbc7d7b9-5x5p9 -- mongosh --quiet --eval "db.version()"
   ```

   The command should output the version of Mongo installed in your pod. This might be different than what is shown here.

   ```
   oc exec mongo-79cbc7d7b9-5x5p9 -- mongosh --quiet --eval "db.version()"
   7.0.8
   ```

Congratulations! You successfully installed the MongoDB server on OpenShift. You are also able to access the server by using the `oc exec` command.

Exercise 7: Songs - Clone the GitHub Repository

Your Tasks

You created a new repository for the songs service from the provided template in a previous lab. If not, go back to the `Creating Get Songs Service with Flask` lab and ensure you complete it before continuing this lab.

1. Open a terminal with `Terminal -> New Terminal` if one is not open already.
2. Change to the project directory. You do not want to clone the songs service in the pictures service directory.

   ```
   cd /home/project
   ```

3. Next, use the export GITHUB_ACCOUNT command to export an environment variable that contains the name of your GitHub account.

   Note: Substitute your real GitHub account for the {your_github_account} placeholder below:

   ```
   export GITHUB_ACCOUNT={your_github_account}
   ```

4. You can use echo the variable in the terminal to double check the value:

   ```
   echo $GITHUB_ACCOUNT
   ```

5. Then use the following commands to clone your repository.

   ```
   git clone https://github.com/$GITHUB_ACCOUNT/Back-End-Development-Songs.git
   ```

6. Change into the devops-capstone-project directory, and execute the `./bin/setup.sh` command.

```
cd Back-End-Development-Songs
bash ./bin/setup.sh
exit
```

7. You should see the follow at the end of the setup execution:

Exercise 8: Songs - Deploy to OpenShift

Let's install the songs microservice to OpenShift. In order to do so, you will simply point OpenShift to your GitHub repository. If you made any changes to the source code, ensure that is committed and pushed back to the main branch of your GitHub repository. Additionally, you will need to tell the application where to find the Mongo server.

Your Tasks

1. Applications within the same OpenShift project can refer to other applications with the name `servicename.openshift_project.svc.cluster.local`. The MongoDb Server application is called `mongo`. Run the following command to get the OpenShift project:

```
oc get project
```

   The result might look as follows:

```
NAME                  DISPLAY NAME   STATUS
sn-labs-captainfedo1                 Active
```

   The project name is `sn-labs-captainfedo1`. Your project name might look different. The full server path becomes `mongo.sn-labs-captainfedo1.svc.cluster.local`.

2. Export your OpenShift project name as the variable OPENSHIFT_PROJECT as follows:

```
export OPENSHIFT_PROJECT=sn-labs-captainfedo1
```

   Replace the value with your OpenShift project.

3. Push the application to RedHat OpenShift using the following command:

```
oc new-app https://github.com/${GITHUB_ACCOUNT}/Back-End-Development-Songs --strategy=source --name=songs --env MONGODB_SERVICE=mongo.${OPENSHIFT_PROJECT}.svc.cluster.local --name songs
```

   - strategy=source: builds directly from source code
   - –name: gives the application a name of `songs`
   - –env: sets the internal address for the `mongo` service

   Take a screenshot of output of the `oc new-app` for the **song microservice**. Save the screenshot as `deploy-getsong-3.jpg` (or .png).

4. The previous step triggers a build for the application. As mentioned in the output, you can trace the logs for the build using this command:

```
oc logs -f buildconfig/songs
```

   You should see an output as follows:

```
Cloning "https://github.com/captainfedoraskillup/private-get-songs" ...
    Commit: 4e2d6e08eaf34c97e4f01a418f90b6e398397c96 (Merge pull request #4 from captainfedoraskillup/openshift)
    Author: captainfedoraskillup <111002162+captainfedoraskillup@users.noreply.github.com>
    Date:   Wed Feb 15 18:35:16 2023 -0800
time="2023-02-23T02:20:19Z" level=info msg="Not using native diff for overlay, this may cause degraded performance for building images: kernel has CONFIG_OVERLAY_FS_REDIRECT_DIR enabled"
I0223 02:20:19.474218       1 defaults.go:102] Defaulting to storage driver "overlay" with options [mountopt=metacopy=on].
Caching blobs under "/var/cache/blobs".
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/python@sha256:7885a4366f211adff41a3ab6b3145ca8fd6e8857c8f6b6a30e43ab3e263498c0...
Getting image source signatures
Copying blob sha256:81051d76ef8269de00d9e37d9aa57e88998c055c18d9a970233ffdb62785c960
Copying blob sha256:bf83366e83795493969367b9608afd8a893bc0719b952a3d190a107bb1b20d2d
Copying blob sha256:a301327116a0e734fb34506202fafa3d0c94b4ec8f4e7e71a73d78c3a117419f
Copying blob sha256:dae34b4e252ee42452b6bf54b0857049ca4115ef4dba7379e2cbd0815017fed8
Copying blob sha256:1593676b86e25227bc00ce535ec5dde8994eff3b65956654b7739043aa7d51ea
Copying config sha256:7caecf9bdc7e25f1928d716918c363637a4eaee8f404d7b5c3e9ef0258999cfe
Writing manifest to image destination
Storing signatures
```

Note that the `-f` flag follows the log. The teminal will not exit automatically. You can use **Ctrl+C** to exit the logs.

5. Once the logs say `Push Successful`, you can go back to the topology and wait for the application to turn green. This may take a couple of minutes.

```
Storing signatures
--> 59b748f50fa
Successfully tagged temp.builder.openshift.io/sn-labs-captainfedo1/songs-1:93ed970d
59b748f50fada2b6f31a5a55c9e0d1cc4c79ddd58f3901ccda0dc4eb243fa3d4
Pushing image image-registry.openshift-image-registry.svc:5000/sn-labs-         captainfedo1/songs:latest ...
Getting image source signatures
Storing signatures
Successfully pushed image-registry.openshift-image-registry.svc:5000/sn-labs-   captainfedo1/songs@sha256:f06d82d7440f550312b948eb51a4fc603c29e0fd7f556f6bb5b09d48113d4323
Push successful
```

6. You can view application logs from the topology by clicking on the `songs` application:

   Click **View logs** to see the logs:
   You should see an output as follows:

7. You can also click the application and it should show a pod running successfully:

8. You need to expose the application so that you can reach it from outside the lab environment. Go back to tab running the lab IDE and open a new terminal. Use the `expose` command as follows:

```
oc expose service/songs
```

   You should see the following output:

```
$ oc expose service/songs
route.route.openshift.io/songs exposed
```

   Take a screenshot of output of the oc expose svc command. Save the screenshot as `deploy-getsong-4.jpg` (or .png).

9. Copy the URL of the application from the application flyout. You will need this URL in the next lab to connect the main Django application with the songs microservice.

   Alternatively, you can use the `oc get route` command in the terminal:

```
oc get route songs
```

   You should see a similar output with the URL listed under the path column:

```
$ oc get route songs
NAME    HOST/PORT                                                                                                                  PATH   SERVICES   PORT       TERMINATION   WILDCARD
songs   songs-sn-labs-captainfedo1.labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud          songs      8080-tcp                 None
```

10. Copy and paste the command from the previous output into a new browser, using the URL `http://$URL/health`.You should get a result as follows:

    Note
    If you get an `Application is not available` error, make sure the URL starts with `http://`, as your browser may redirect you to the `https://` secure version.

```
 http://songs-sn-labs-captainfedo1.labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.appdomain.cloud/health
```

```
{
  "status": "OK"
}
```
Take a screenshot of output of the hiting the `/health` endpoint. Save the screenshot as `deploy-getsong-5.jpg` (or .png).

Evidence

1. Take a screenshot of output of the `oc new-app` for the **song microservice**. Save the screenshot as `deploy-getsong-3.jpg` (or .png).
2. Take a screenshot of output of the oc expose svc command. Save the screenshot as `deploy-getsong-4.jpg` (or .png).
3. Take a screenshot of the running application in your browser with the `/health` endpoint. Save the screenshot as `deploy-getsong-5.jpg` (or .png).

   If you don't get a result back or run into an error, you can follow these steps again or ask for help in the course forum.

Congratulations! You just deployed the Songs microservice on RedHat OpenShift.

- Ensure you copy the public URLs of both services. You will need it for the final lab of this capstone project.
- Please do not logout or close this lab. The Code Engine instance running here is required for the 2 Public URLs to give the correct output in the final lab of this capstone project.

Author(s)

Lavanya T S
CF