Lab: Deploy to Kubernetes

Estimated time needed: 60 minutes

Welcome to the **Deploy to Kubernetes** hands-on lab. Now that your microservice has been built and tested, it is time to deploy it to a Kubernetes environment to run it. In particular, you will use OpenShift, which is based on Kubernetes and adds additional developer capabilities.

Objectives

In this lab, you will:

- Take the next story from the Sprint Backlog to work on
- Create a Dockerfile and build an image from your microservice
- Create Kubernetes manifests for your deployment
- Deploy your Docker image in an OpenShift Kubernetes cluster
- View the logs to ensure your service is running
- Make a pull request and merge your changes
- Move the story to Done

Note: Important Security Information

Welcome to the Cloud IDE with OpenShift. This is where all your development will take place. It has all the tools you will need to use Docker for deploying a PostgreSQL database.

It is important to understand that the lab environment is **ephemeral**. It only lives for a short time before it is destroyed. It is imperative that you push all changes made to your own GitHub repository so that it can be recreated in a new lab environment any time it is needed.

Also note that this environment is shared and therefore not secure. You should not store any personal information, usernames, passwords, or access tokens in this environment for any purpose.

Your Task

1. If you haven't generated a **GitHub Personal Access Token** you should do so now. You will need it to push code back to your repository. It should have `repo` and `write` permissions and be set to expire in `60` days. When Git prompts you for a password in the Cloud IDE environment, use your Personal Access Token instead.

2. The environment may be recreated at any time, so you may find that you have to perform the **Initialize Development Environment** each time the environment is created.

Note on Screenshots

Throughout this lab, you will be prompted to take screenshots and save them on your device. These will be required either to answer graded quiz questions or for submission under **Option 1: AI-Graded Submission and Evaluation** or **Option 2: Peer-Graded Submission and Evaluation** at the end of this course. Your screenshot must have either the .jpeg or .png extension.

To take screenshots, you can use various free screen-capture tools or your operating system's shortcut keys. For example:

- **Mac:** You can use `Shift + Command + 3` (⇧ + ⌘ + 3) on your keyboard to capture your entire screen, or `Shift + Command + 4` (⇧ + ⌘ + 4) to capture a window or area. They will be saved as a file on your desktop.

- **Windows:** You can capture your active window by pressing `Alt + Print Screen` on your keyboard. This command copies an image of your active window to the clipboard. Next, open an image editor, paste the image from your clipboard to the image editor, and save the image.

Initialize Development Environment

Because the Cloud IDE with OpenShift environment is ephemeral, it may be deleted at any time. The next time you come into the lab, a new environment may be created. Unfortunately, this means that you will need to initialize your development environment every time it is recreated. This shouldn't happen too often, as the environment can last for several days at a time, but when it is removed, this is the procedure to recreate it.

Overview

Each time you need to set up your lab development environment, you will need to run three commands.

Each command will be explained in further detail, one at a time, in the following section.

`{your_github_account}` represents your GitHub account username.

The commands include:

```
git clone https://github.com/{your_github_account}/devops-capstone-project.git
cd devops-capstone-project
bash ./bin/setup.sh
exit
```

Now, let's discuss each of these commands and explain what needs to be done.

Task Details

Initialize your environment using the following steps:

1. Open a terminal with `Terminal` -> `New Terminal` if one is not already open.
2. Next, use the `export GITHUB_ACCOUNT=` command to export an environment variable that contains the name of your GitHub account.

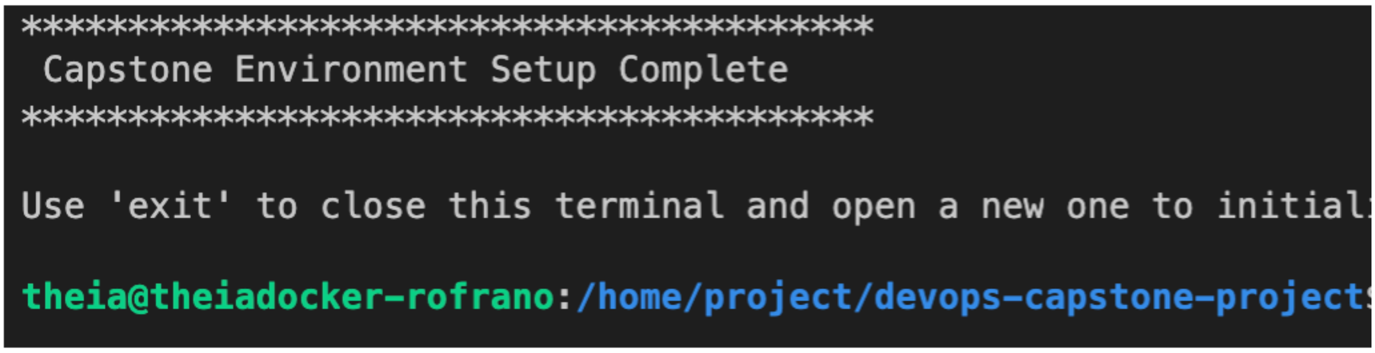
   Note: Substitute your real GitHub account for the `{your_github_account}` placeholder below:

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

3. Then use the following commands to clone your repository, change into the `devops-capstone-project` directory, and execute the `./bin/setup.sh` command.

   ```
   git clone https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git
   cd devops-capstone-project
   bash ./bin/setup.sh
   ```

   You should see the following at the end of the setup execution:

   ```
   ****************************************
    Capstone Environment Setup Complete
   ****************************************

   Use 'exit' to close this terminal and open a new one to initial

   theia@theiadocker-rofrano:/home/project/devops-capstone-project
   ```

4. Finally, use the `exit` command to close the current terminal. The environment will not be fully active until you open a new terminal in the next step.

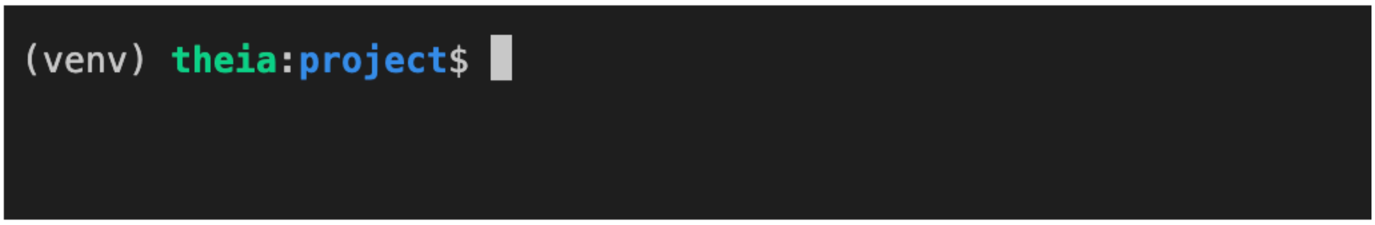
   ```
   exit
   ```

Validate

In order to validate that your environment is working correctly, you must open a new terminal because the Python virtual environment will only activate when a new terminal is created. You should have ended the previous task by using the `exit` command to exit the terminal.

1. Open a terminal with `Terminal` -> `New Terminal` and check that everything worked correctly by using the `which python` command:

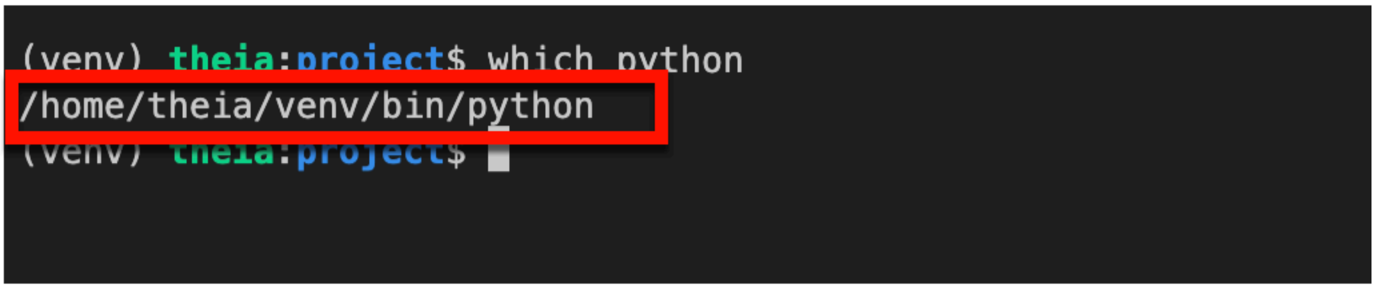
   Your prompt should look like this:


```
(venv) theia:project$
```

Check which Python you are using:

```
which python
```

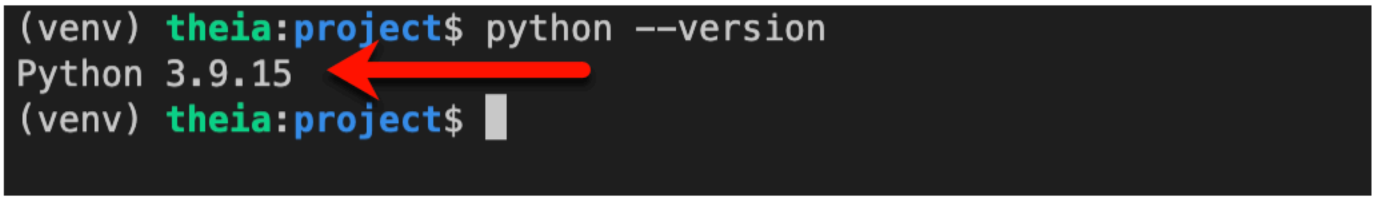
You should get back:


```
(venv) theia:project$ which python
/home/theia/venv/bin/python
(venv) theia:project$
```

Check the Python version:

```
python --version
```

You should get back some patch level of Python 3.9:


```
(venv) theia:project$ python --version
Python 3.9.15
(venv) theia:project$
```

This completes the setup of the development environment. Any time your environment is recreated, you will need to follow this procedure.

You are now ready to start working.

Exercise 1: Pick Up the First Story

The first thing you need to do is go to your kanban board to get a story to work on. Take the first story from the top of the Sprint Backlog, move it to In Progress, assign it to yourself, and read the contents.

Your Task

1. Go to your kanban board and take the first story from the top of the `Sprint Backlog`. It should be titled "*Containerize your microservice using Docker*".
2. Move the story to `In Progress`.
3. Open the story and assign it to *yourself*.
4. Read the contents of the story.

Results

The story should look similar to this:

Containerize your microservice using Docker

As a developer
I need to containerize my microservice using Docker
So that I can deploy it easily with all of its dependencies

Assumptions

- Create a `Dockerfile` for repeatable builds
- Use a `Python:3.9-slim` image as the base
- It must install all of the Python requirements
- It should not run as `root`
- It should use the `gunicorn` wsgi server as an entry point

Acceptance Criteria

```
Given the Docker image named accounts has been created
When I use `docker run accounts`
Then I should see the accounts service running in Docker
```

You are now ready to begin working on your story.

Exercise 2: Create a Dockerfile

In reading your story you see that the assumptions state that you must create a `Dockerfile` with the following attributes:

- Create a `Dockerfile` for repeatable builds
- Use a `Python:3.9-slim` image as the base
- It must install all of the Python requirements
- It should not run as `root`
- It should use the `gunicorn` wsgi server as an entry point

Let's take these in order.

Your Task

1. Change to your project directory: `cd devops-capstone-project`.

2. Use the `git checkout -b add-docker` command to create a new branch called `add-docker` to work on in the development environment.

3. Run `nosetests` and make sure that all of the test cases are passing. Fix any failing tests before proceeding.

4. In the root of the repository, create a file named `Dockerfile`.

5. Edit the `Dockerfile` and start it `FROM` the `python:3.9-slim` image.

   ▼ Click here for the answer.

   ```
   FROM python:3.9-slim
   ```

6. Establish a `WORKDIR` of `/app`, `COPY` the `requirements.txt` file into the working directory in the image, and `RUN` the `pip` command to install the requirements using the `--no-cache-dir` option to keep the image small.

   ▼ Click here for the answer.

   ```
   # Create working folder and install dependencies
   WORKDIR /app
   COPY requirements.txt .
   RUN pip install --no-cache-dir -r requirements.txt
   ```

7. Copy the `service` package into the working directory of the same name in the image.

   ▼ Click here for the answer.

   ```
   # Copy the application contents
   COPY service/ ./service/
   ```

8. Create a non-root user called `theia`, change the ownership of the `/app` folder recursively to `theia`, and switch to the `theia` user.

   ▼ Click here for the answer.

   ```
   # Switch to a non-root user
   RUN useradd --uid 1000 theia && chown -R theia /app
   USER theia
   ```

9. Finally, `EXPOSE` port `8080` and create a `CMD` statement that runs: `gunicorn --bind=0.0.0.0:8080 --log-level=info service:app`

   ▼ Click here for the answer.

   ```
   # Run the service
   EXPOSE 8080
   CMD ["gunicorn", "--bind=0.0.0.0:8080", "--log-level=info", "service:app"]
   ```

Results

You can check that your `Dockerfile` looks like the following:

▼ Click here to check your work.

```
FROM python:3.9-slim
# Create working folder and install dependencies
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# Copy the application contents
COPY service/ ./service/
# Switch to a non-root user
RUN useradd --uid 1000 theia && chown -R theia /app
USER theia
# Run the service
EXPOSE 8080
CMD ["gunicorn", "--bind=0.0.0.0:8080", "--log-level=info", "service:app"]
```

Exercise 3: Create a Docker Image

Now that you have created a `Dockerfile`, it's time to create an image from it to see if it works.

Your Task

1. Open a terminal and use the `docker build` command to build a Docker image called `accounts` from the `Dockerfile`.

   ▼ Click here for the answer.

```
docker build -t accounts .
```

2. Use the `docker run` command to test that your image works properly. The PostgreSQL database is running in a Docker container named `postgres` so you will need to `--link postgres` and set the environment variable `DATABASE_URI` to point to it. You might also want to use the `--rm` flag to remove the container when it exists.

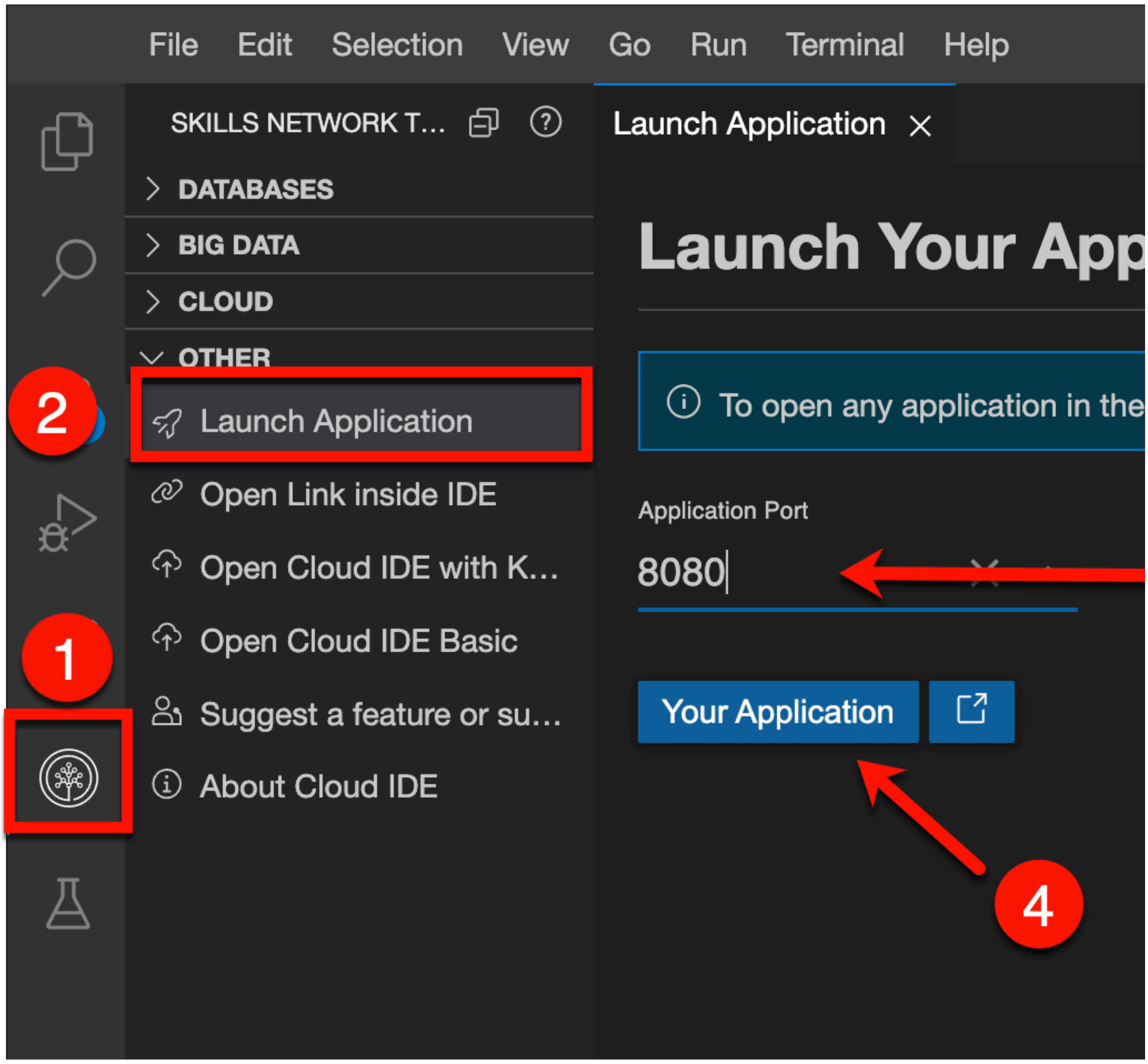
   If it worked, you should see the message:

   ```
   ... [INFO] [__init__] Service initialized!
   ```

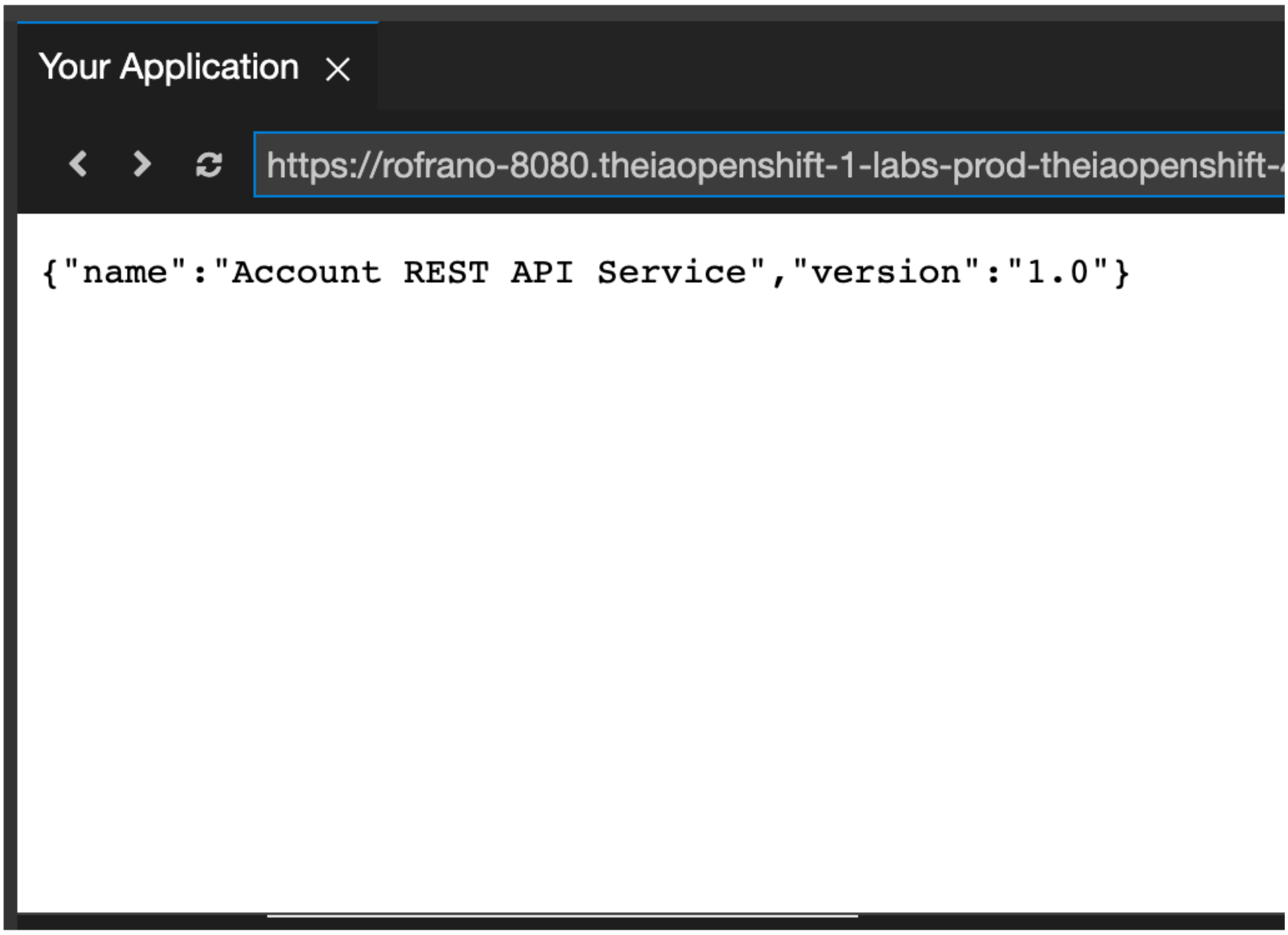
▼ Click here for the answer.

```
docker run --rm \
    --link postgresql \
    -p 8080:8080 \
    -e DATABASE_URI=postgresql://postgres:postgres@postgresql:5432/postgres \
    accounts
```

3. Check that your application is running by (1) clicking the `Skills Network` icon, (2) selecting `Other` -> `Launch Application`, (3) entering an Application Port of `8080`, and (4) clicking the `Your Application` button.



You should see the following:


Your Application
https://rofrano-8080.theiaopenshift-1-labs-prod-theiaopenshift-

```
{"name":"Account REST API Service","version":"1.0"}
```

4. Use `Ctrl+C` to stop your container.
5. Tag the image as `us.icr.io/$SN_ICR_NAMESPACE/accounts:1` and push it to the IBM Cloud registry.

    Note: The environment variable `SN_ICR_NAMESPACE` contains your image namespace in the IBM Cloud Container Registry.

▼ Click here for the answer.

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

Evidence

For Option 1: AI-Graded Submission and Evaluation

After launching your application in the internal web browser on port **8080**, copy the **JSON output** and save it in a file named **`kube-app-output`** for submission.

For Option 2: Peer-Graded Submission and Evaluation

After launching your application in the internal web browser, take a **screenshot** of the application output and save it as `kube-app-output.jpg` or `kube-app-output.png`.

Exercise 4: Make a Pull Request

Now that you have a working Docker image, it's time to push the `Dockerfile` up to GitHub and make a pull request, merge the request, and move your story to `Done`.

Your Task

1. Use `git status` to make sure that you have committed your changes locally in the development environment.
2. Use the `git add` command to add the new `Dockerfile` to the staging area.
3. Commit your changes using the message `Added docker support`.

4. Push your local changes to a remote branch.

    Note: Use your GitHub **Personal Access Token** as your password in the Cloud IDE environment. You may also need to configure Git the first time you use it with:

    ```
    git config --local user.email "you@example.com"
    git config --local user.name "Your Name"
    ```

▶ Click here for the answer.

5. Make a pull request, which should kick off the GitHub Actions that are now enabled on your repository.

6. Once the test cases pass, merge your pull request.

7. Move your story to the `Done` column on your kanban board.

8. Pull the last code down to your development environment and delete your old branch.

    ```
    git checkout main
    git pull
    git branch -d add-docker
    ```

Evidence

For both **Option 1: AI-Graded Submission and Evaluation** and **Option 2: Peer-Graded Submission and Evaluation**,
open your **Kanban board** and take a screenshot showing the story **"Containerize your microservice using Docker"** moved to the **Done** column and save the screenshot as `kube-docker-done.jpeg` or `kube-docker-done.png`.

Exercise 5: Pick Up the Next Story

It's now time to go to your kanban board to get the next story to work on. It should be at the top of the Sprint Backlog.

Your Task

1. Go to your kanban board and take the next story from the top of the `Sprint Backlog`. It should be titled "*Deploy your Docker image to Kubernetes*".

2. Move the story to `In Progress`.

3. Open the story and assign it to *yourself*.

4. Read the contents of the story.

Results

The story should look similar to this:

Deploy your Docker image to Kubernetes

As a service provider
I need my service to run on Kubernetes
So that I can easily scale and manage the service

Assumptions

- Kubernetes manifests will be created in yaml format
- These manifests could be useful to create a CD pipeline
- The actual deployment will be to OpenShift

Acceptance Criteria

```
Given the Kubernetes manifests have been created
When I use the oc command to apply the manifests
Then the service should be deployed and run in Kubernetes
```

You are now ready to begin working on your second story.

Exercise 6: Deploy to Kubernetes

For the "Deploy to Kubernetes" story, you must create the manifests required to consistently deploy your microservice. At some point in the future, you will need to create a CD pipeline to perform continuous delivery, so while you are deploying manually now, it's important that you create manifests that can be used later in the pipeline.

You are going to need a PostgreSQL database in Kubernetes for your application to use. Luckily, your are using OpenShift, which comes with a number of templates for creating services. Your first task is to deploy the `postgresql-ephemeral` template, which will create an ephemeral PostgreSQL database with no backing storage for test purposes.

Your Task

1. Use the `git checkout -b add-kubernetes` command to create a new branch called `add-kubernetes` to work on in the development environment.
2. Use the below command to define and create resources based on the `postgresql-ephemeral` JSON template.

   ```
   oc create -f postgresql-ephemeral-template.json
   ```

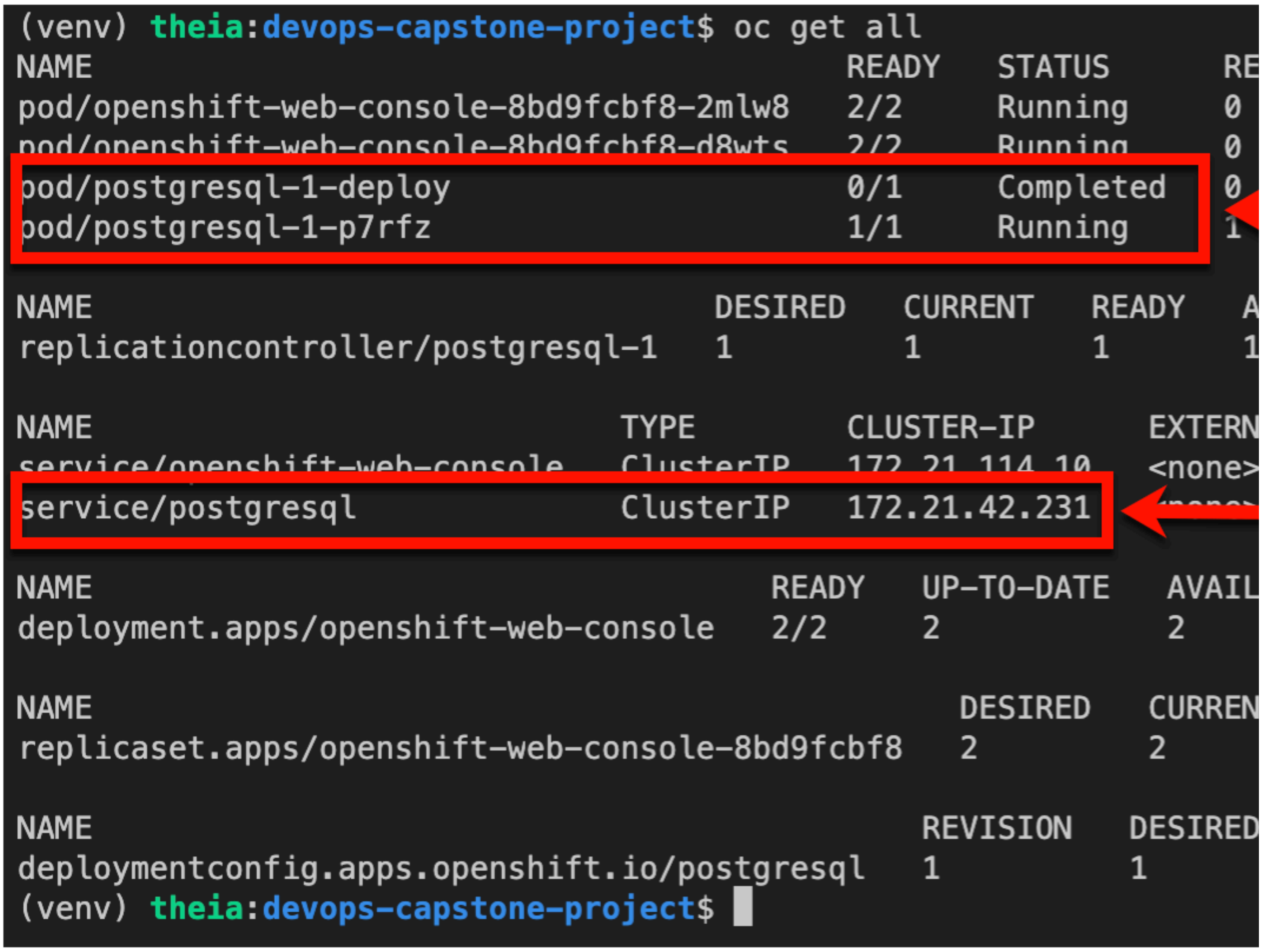
3. Use the `oc new-app` command to deploy the `postgresql-ephemeral` instance based on the template.

   ▼ Click here for the answer.

   ```
   oc new-app postgresql-ephemeral
   ```

4. Use `oc get all` to make sure that the postgres service is defined and the postgres pod is running.

Results



```
(venv) theia:devops-capstone-project$ oc get all
NAME                                          READY   STATUS      RE
pod/openshift-web-console-8bd9fcbf8-2mlw8     2/2     Running     0
pod/openshift-web-console-8bd9fcbf8-d8wts     2/2     Running     0
pod/postgresql-1-deploy                       0/1     Completed   0
pod/postgresql-1-p7rfz                        1/1     Running     1

NAME                                   DESIRED   CURRENT   READY   A
replicationcontroller/postgresql-1    1         1         1       1

NAME                              TYPE        CLUSTER-IP      EXTERN
service/openshift-web-console     ClusterIP   172.21.114.10   <none>
service/postgresql                ClusterIP   172.21.42.231   <none>

NAME                                    READY   UP-TO-DATE   AVAIL
deployment.apps/openshift-web-console   2/2     2            2

NAME                                               DESIRED   CURREN
replicaset.apps/openshift-web-console-8bd9fcbf8    2         2

NAME                                               REVISION   DESIRED
deploymentconfig.apps.openshift.io/postgresql      1          1
(venv) theia:devops-capstone-project$ 
```

You are now ready to create Kubernetes manifests for your microservice.

Exercise 7: Create Manifests

Here is a tip for getting started creating manifest yaml files. You can use the `kubectl` or `oc` CLI to create a deployment or service and capture the definition in a yaml file by adding the flags `--dry-run=client -o yaml`. This code doesn't actually create anything (`--dry-run=client`) but sends output to yaml (`-o yaml`). Then all you need to do is redirect that to a file.

1. Create a manifest definition for the account deployment using the `oc create deployment` command with the `--dry-run -o yaml` option and redirect it to a file called `deploy/deployment.yaml`. Specify the image that you pushed to the IBM Cloud registry and request three replicas.

   ▼ Click here for the answer.

   ```
   oc create deployment accounts \
     --image=us.icr.io/$SN_ICR_NAMESPACE/accounts:1 \
     --replicas=3 \
     --dry-run=client -o yaml > deploy/deployment.yaml
   ```

2. Your microservice needs to know the details about the postgres database that you just deployed. In particular, it needs the following environment variables: `DATABASE_HOST`, `DATABASE_NAME`, `DATABASE_PASSWORD`, and `DATABASE_USER`. Use the `oc describe` command to see what keys are in the secret that you can use:

   ▼ Click here for the answer.

   ```
   oc describe secret postgresql
   ```

```
...
Data
====
database-name:      8 bytes
database-password:  16 bytes
database-user:      7 bytes
```

3. Edit the `deploy/deployment.yaml` file and use the keys that you found in the secret along with a `DATABASE_HOST` of `postgresql` to add the required environment variables to the manifest.

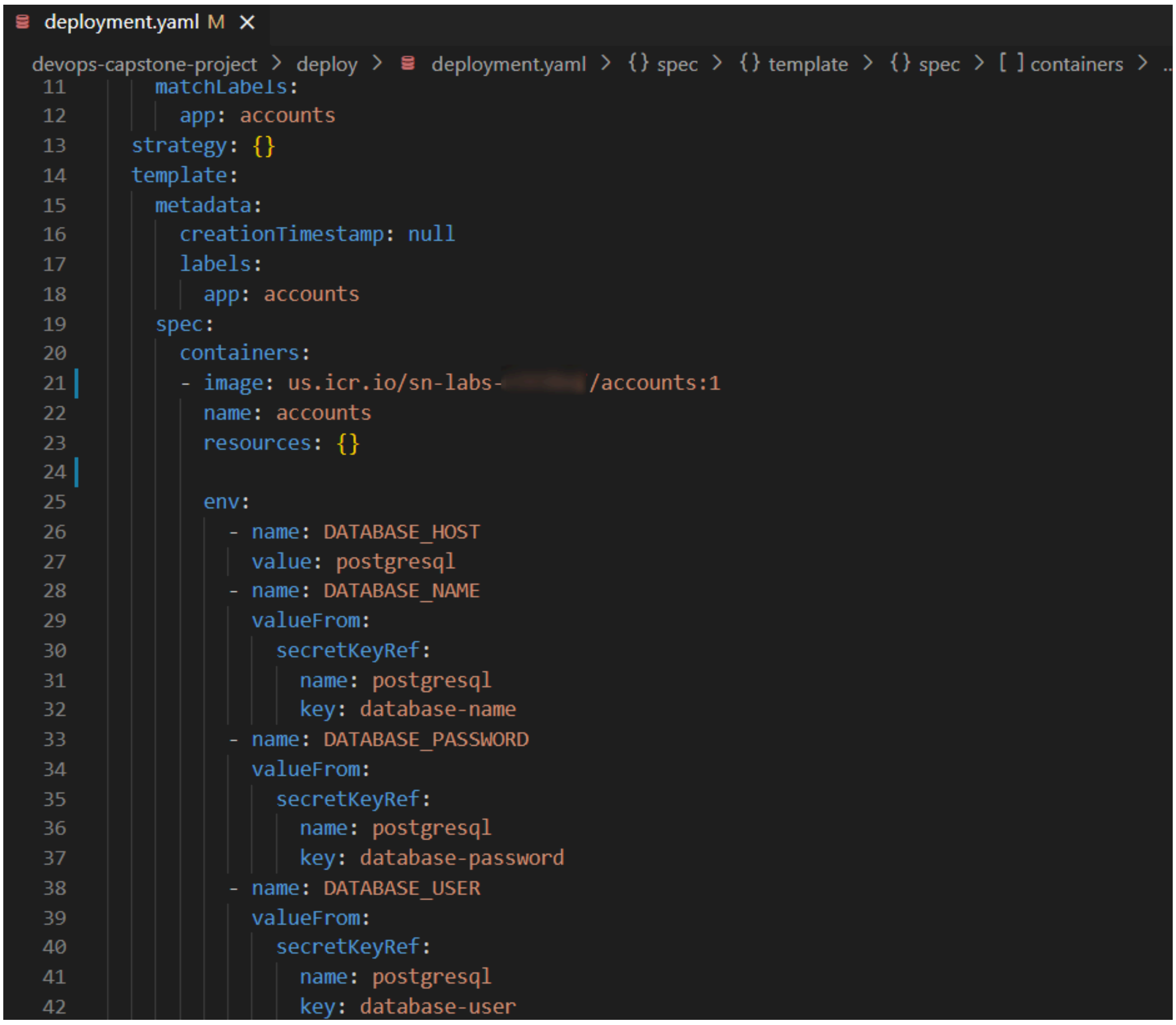
   ▼ Click here for a hint.
   You can create an environment variable from a secret key like this:

   ```
   env:
    - name: DATABASE_NAME
        valueFrom:
        secretKeyRef:
            name: postgresql
            key: database-name
   ```

   ▼ Click here for the answer.

   ```
         env:
           - name: DATABASE_HOST
             value: postgresql
           - name: DATABASE_NAME
             valueFrom:
               secretKeyRef:
                 name: postgresql
                 key: database-name
           - name: DATABASE_PASSWORD
             valueFrom:
               secretKeyRef:
                 name: postgresql
                 key: database-password
           - name: DATABASE_USER
             valueFrom:
               secretKeyRef:
                 name: postgresql
                 key: database-user
   ```

Results


deployment.yaml M ×
devops-capstone-project > deploy > deployment.yaml > {} spec > {} template > {} spec > [ ] containers > ..

```yaml
    matchLabels:
      app: accounts
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: accounts
    spec:
      containers:
      - image: us.icr.io/sn-labs-[redacted]/accounts:1
        name: accounts
        resources: {}

        env:
          - name: DATABASE_HOST
            value: postgresql
          - name: DATABASE_NAME
            valueFrom:
              secretKeyRef:
                name: postgresql
                key: database-name
          - name: DATABASE_PASSWORD
            valueFrom:
              secretKeyRef:
                name: postgresql
                key: database-password
          - name: DATABASE_USER
            valueFrom:
              secretKeyRef:
                name: postgresql
                key: database-user
```

1. Apply this deployment using `oc create` and point it to your `deploy/deployment.yaml` file.

   ▼ Click here for the answer.

   ```
   oc create -f deploy/deployment.yaml
   ```

2. Create a manifest definition for the account service using the `oc expose` command, using a `type` of `NodePort` and a `Port` of `8080`, and redirect it to a file called `deploy/service.yaml`.

   ▼ Click here for the answer.

   ```
   oc expose deploy accounts \
     --type=ClusterIP \
     --port=8080 \
     --dry-run=client -o yaml > deploy/service.yaml
   ```

3. Apply this service using the `oc create` command and point it to your `deploy/service.yaml` file.

   ▼ Click here for the answer.

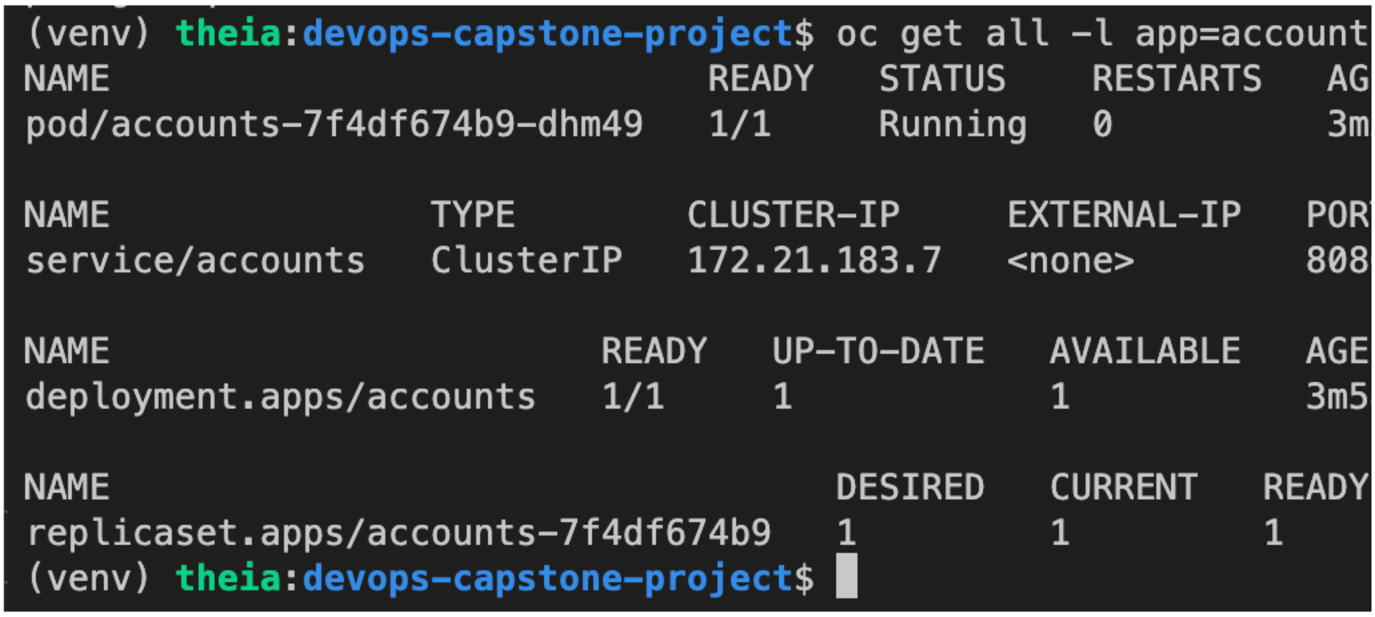
   ```
   oc create -f deploy/service.yaml
   ```

4. Now it's time to see if everything is running. Use `oc get all` and filter by the level `app=accounts` to see your deployment running.

   ▼ Click here for the answer.

   ```
   oc get all -l app=accounts
   ```

   You should see something similar to the following:



```
(venv) theia:devops-capstone-project$ oc get all -l app=account
NAME                              READY   STATUS    RESTARTS   AG
pod/accounts-7f4df674b9-dhm49     1/1     Running   0          3m

NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   POR
service/accounts   ClusterIP   172.21.183.7    <none>        808

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/accounts      1/1     1            1           3m5

NAME                                   DESIRED   CURRENT   READY
replicaset.apps/accounts-7f4df674b9    1         1         1
(venv) theia:devops-capstone-project$ 
```

   Note: There should be a `deployment`, `replicaset`, `pod`, and `service`.

5. Finally, expose your service using an OpenShift route. Use the `oc create` command to create a `route` called `accounts` with `edge` termination that exposes the `--service` named `accounts`.

   ▼ Click here for the answer.

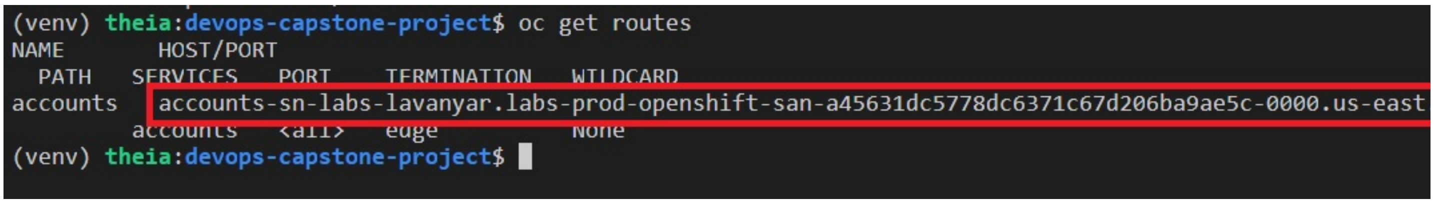
   ```
   oc create route edge accounts --service=accounts
   ```

6. Use the `oc get routes` command to get the route that was assigned to your service.

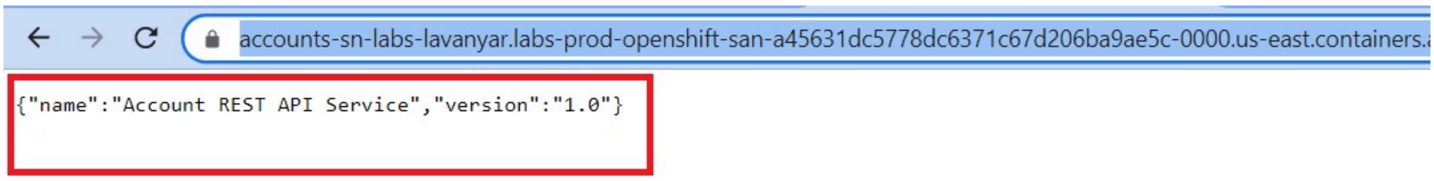
   ```
   oc get routes
   ```

7. Copy the URL of your route and paste it into your browser to see your application running in OpenShift.

Results



```
(venv) theia:devops-capstone-project$ oc get routes
NAME        HOST/PORT
  PATH    SERVICES    PORT     TERMINATION    WILDCARD
accounts   accounts-sn-labs-lavanyar.labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east
         accounts    <all>    edge           None
(venv) theia:devops-capstone-project$
```



```
accounts-sn-labs-lavanyar.labs-prod-openshift-san-a45631dc5778dc6371c67d206ba9ae5c-0000.us-east.containers.
{"name":"Account REST API Service","version":"1.0"}
```

Exercise 8: Make Another Pull Request

Now that you have a working deployment, it's time to push the Kubernetes manifests up to GitHub, make a pull request, merge the request, and move your story to `Done`.

Your Task

1. Use `git status` to make sure that you have committed your changes locally in the development environment.
2. Use the `git add` command to add the new `deployment.yaml` and `service.yaml` to the staging area.
3. Commit your changes using the message `Added Kubernetes support`.
4. Push your local changes to a remote branch.

    Note: Use your GitHub **Personal Access Token** as your password in the Cloud IDE environment.You may need to configure Git before using it for the first time with the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

This sets your name and email, which will be associated with your commits.

1. Make a pull request on GitHub to merge your changes into the `main` branch. Also, check if it kicks off the GitHub Action that is now enabled on your repository.
2. Once the test cases pass, merge your pull request.
3. Move your story to the `Done` column on your kanban board.
4. Pull the latest code down to your development environment and delete your old branch.

```
git checkout main
git pull
git branch -d add-kubernetes
```

Evidence

For both **Option 1: AI-Graded Submission and Evaluation** and **Option 2: Peer-Graded Submission and Evaluation**, you must provide a screenshot showing the story marked as complete.

Open your Kanban board and take a screenshot showing the story **"Deploy your Docker image to Kubernetes"** in the **Done** column.
Save the screenshot as `kube-kubernetes-done.jpeg` or `kube-kubernetes-done.png`.

Collect Final Evidence

You need to collect the following evidence as proof of the completion of this lab.

For **Option 1: AI-Graded Submission and Evaluation**, you must collect and submit the following evidence:

1. Open your `Dockerfile` on GitHub and copy the `public URL link`.
   You will submit this URL as part of your evidence.

2. Run the following command in your terminal:

   ```
   docker image ls
   ```

   Copy and save the complete terminal output text in a file and save it as `kube-images`.

3. Run the following command:

   ```
   oc get all -l app=accounts
   ```

   Take a screenshot of the output and save it as `kube-deploy-accounts.jpeg` or `kube-deploy-accounts.png`.

For **Option 2: Peer-Graded Submission and Evaluation**, you must collect and submit the following evidence:

1. Open your `Dockerfile` on GitHub and copy the `public URL link`.
   You will submit this URL as part of your evidence.

2. Run the following command in your terminal:

   ```
   docker image ls
   ```

   Take a screenshot of the output and save it as `kube-images.jpeg` or `kube-images.png`.

3. Run the following command:

   ```
   oc get all -l app=accounts
   ```

   Take a screenshot of the output and save it as `kube-deploy-accounts.jpeg` or `kube-deploy-accounts.png`.

Conclusion

Congratulations! You have built a Docker image from a `Dockerfile` and deployed that image to an OpenShift Kubernetes cluster using yaml manifests that can be reused in a continuous delivery (CD) pipeline.

Next Steps

Implement the third story in Sprint 3.

Author

John J. Rofrano

Other Contributor(s)