Peer Review Assignment - Developers

Estimated time needed: **60** minutes

Objectives

In this assignment you will:

- Use Watson APIs to create functions.
- Create a function that translates English to French.
- Create a function that translates French to English.
- Run coding standards check against the functions above.
- Write unit tests to test the above functions.
- Run unit tests and interpret the results.
- Package the above functions and tests as a standard python package.

> **Important Notice** - Please keep in mind that sessions for this lab environment are not persisted. If you will not be completing the entire lab in one sitting, please save your code outisde of this environment, so you can resume your work without loss.

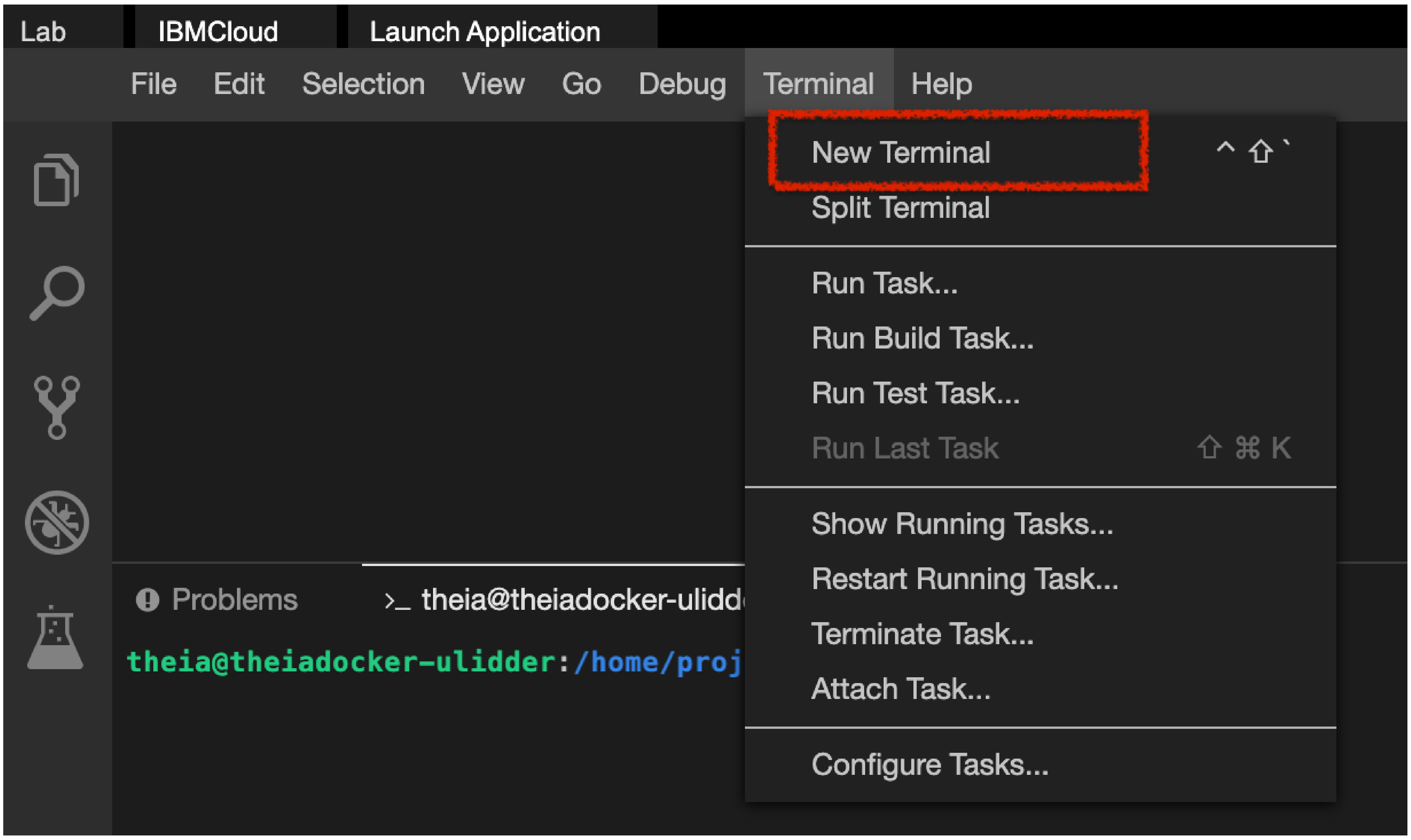
▶ 👉 *Click here for instructions to save your code on github*

Provision Watson Translation Service

- Before you start, ensure you have provisioned an instance of the Watson Language Translator service and have API information available.

Task1: Write a function that translates English text to French in translator.py

1. Open a terminal window by using the menu in the editor: Terminal > New Terminal.



```
theia@theiadocker-ulidder:/home/proj
```

2. Go to the project home directory.

```
cd /home/project
```

3. Run the following command to Git clone the project directory from the clone URL you had copied in the prework lab.

```
[ ! -d 'xzceb-flask_eng_fr' ] && git clone <paste_your_repo_name>
```

4. Change to the `final_project` folder.

```
cd /home/project/xzceb-flask_eng_fr/final_project
```

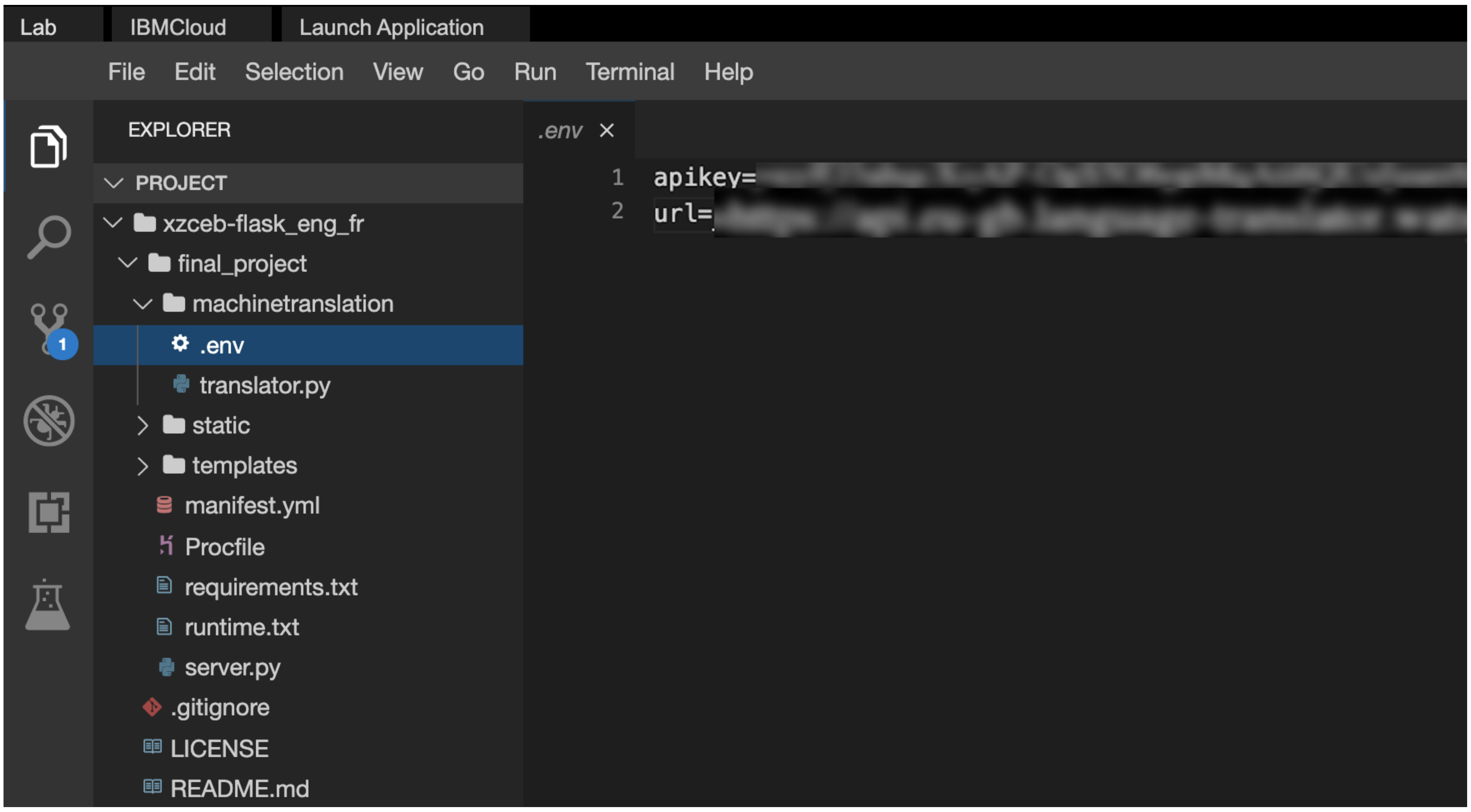
5. Create folder named `machinetranslation` and change to that directory.

```
mkdir machinetranslation
cd machinetranslation
```

6. Install the packages that you will be using in this code, namely `ibm_watson` and `python-dotenv`.

```
pip3 install -q PyJWT==1.7.1 flask-appbuilder==3.4.4 markupsafe==1.1.1 httpx==0.20 Flask ibm-watson python-dotenv
```

7. Create a file `.env` under the `machinetranslation` directory, which maps the apikey and url of the Language Translator Service that you created in the IBM Cloud.



```
xzceb-flask_eng_fr
  final_project
    machinetranslation
      .env
      translator.py
    static
    templates
    manifest.yml
    Procfile
    requirements.txt
    runtime.txt
    server.py
  .gitignore
  LICENSE
  README.md
```

```
apikey=[redacted]
url=[redacted]
```

▶ 👉 Click here to see where the credentials can be copied from

8. In the explorer, go to the `machinetranslation` directory and create a new file called `translator.py`. Enter the following lines of code.

```
import json
from ibm_watson import LanguageTranslatorV3
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
import os
from dotenv import load_dotenv

load_dotenv()

apikey = os.environ['apikey']
url = os.environ['url']
```

9. Add code to create an instance of the IBM Watson Language translator in `translator.py`.

> Hint : You may refer to the IBM Watson Language Translation API documents here.

> 📷 Take a screenshot of your functions and save it as a .jpg or .png with the filename `translator_instance`. You will be prompted to upload the screenshot in the Peer Assignement that follows.

10. Add function **englishToFrench** which takes in the `englishText` as a string argument, in `translator.py`. Use the instance of the Language Translator you created previously, to translate the text input in English to French and return the French text.

```
def englishToFrench(englishText):
    #write the code here
    return frenchText
```

📷 Take a screenshot of your functions and save it as a .jpg or .png with the filename `e2f_translator_function`. You will be prompted to upload the screenshot in the Peer Assignement that follows.

11. Add function **frenchToEnglish** which takes in the `frenchText` as a string argument, in `translator.py`. Use the instance of the Language Translator you created previously, to translate the text input in French to English and return the English text.

```
def frenchToEnglish(frenchText):
    #write the code here
    return englishText
```

> 📷 Take a screenshot of your functions and save it as a .jpg or .png with the filename `f2e_translator_function`. You will be prompted to upload the screenshot in the Peer Assignement that follows.

Task 2: Write the unit tests for English to French translator and French to English translator function in tests.py

1. Create a new file called `tests.py` in the `machinetranslation` directory.
2. Write at least 2 tests in `tests.py` for each method
3. Test for null input for `frenchToEnglish`
4. Test for null input for `englishToFrench`.
5. Test for the translation of the world 'Hello' and 'Bonjour'.
6. Test for the translation of the world 'Bonjour' and 'Hello'.
7. Take a screenshot of your unit tests and save it as a .jpg or .png with the filename `translation_unittests`.

Task 3: Check your code against python coding standards

1. Run `pylint translator.py` to check the coding standard compliance in your code. Refer to this exercise you did earlier, if needed.
2. Make sure your rating is at least 7.
3. 📷 Take a screenshot of the output of the pylint analysis report showing your score and save it as a .jpg or .png with the filename `pylint_score`.

Task 4: Run tests

1. At the terminal run the command

```
python3 tests.py
```

2. 📷 Take a screenshot of test results and save it as a .jpg or .png with the filename **unit_test_results**.

Task 5: Package the above functions and tests as a standard python package.

1. Create `__init__.py` file in the directory `machinetranslation`.

2. Create a folder called `tests` under the newly created folder

3. Copy the unit `tests` into the tests folder

> 📷 Take a screenshot of the folder structure of the package (From the menu go to View -> Explorer to set the explorer view) and save it as a .jpg or .png with the filename `package_folder_structure`.

Task 6: Import the package into server.py and create flask end points

1. Import the package `machinetranslation` in server.py.

2. In the server.py, for end-point `/`, implement a method that renders the `index.html`.

3. In the space provided in server.py for end-point `/englishToFrench` implement a method that uses the appropriate translation function through the package you created in the previous part. The function should take the English text as input through the request parameter and return a string.

4. In the space provided in server.py for end-point `/frenchToEnglish` implement a method that uses the appropriate translation function through the package you created in the previous part. The function should take the French text as input through the request parameter and return a string.

5. Push the code to GitHub.

▶ 👉 *Click here for instructions to push your code to github*

Task 7: Run the server

1. Change to the project directory and run the server from your terminal.

```
cd /home/project/xzceb-flask_eng_fr/final_project && python3 server.py
```

2. You will see that the server starts up in port 8080.

3. Click on `Launch Application` from the menu and connect to port 8080.

4. A new browser window opens up with the index page.

    - 📷 Take a screen shot of the translation from English to French
    - 📷 Take a screen shot of the translation from French to English

Task 8: (Optional) Deploy the application on IBM Cloud.

> Note : This will only work if Cloud Foundry is available.

1. Change to the `final_project` directory.

    ```
    cd /home/projects/xzceb-flask_eng_fr/final_project
    ```

2. Login to the ibmcloud with your email and enter your password when prompted.

    ```
    ibmcloud login -u <your email>
    ```

3. Set the target region and owner. Run the followinf command after you replace `REGION` with your region and the `OWNER` with your email id.

    ```
    ibmcloud target --cf-api https://api.REGION.cf.cloud.ibm.com -r REGION -o OWNER
    ```

4. Create an account space called `translator`.

    ```
    ibmcloud account space-create translator
    ```

5. Target the account space you just created.

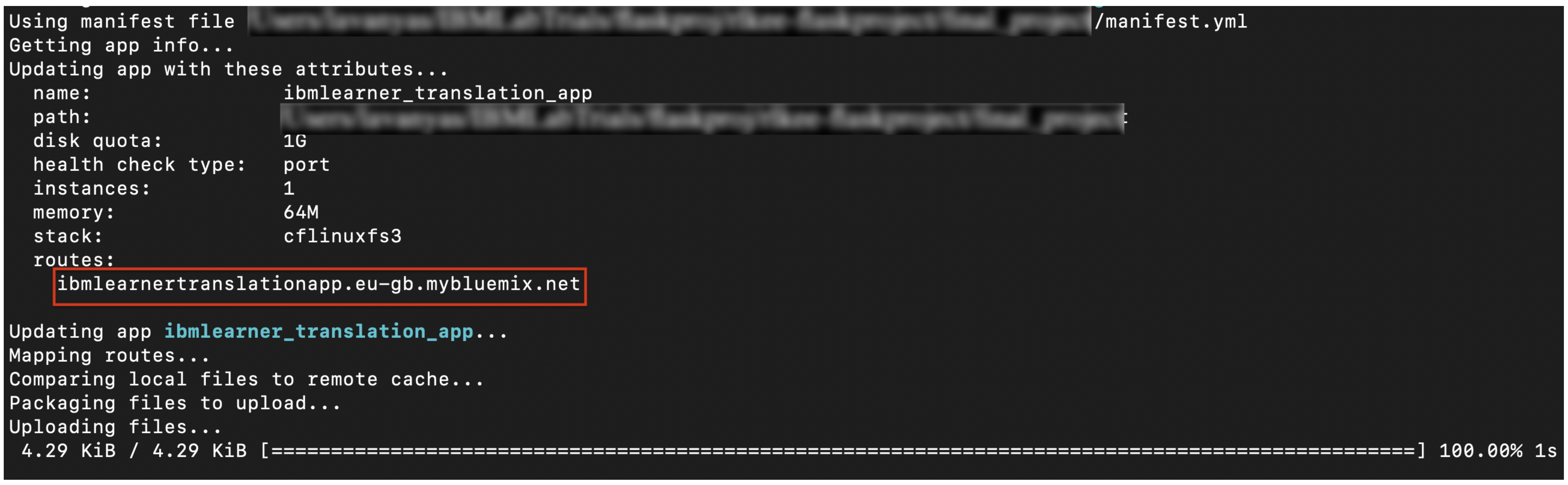
    ```
    ibmcloud target -s translator
    ```

6. Replace line 3 in the `manifest.yml` to give a name for your web application.

7. Deploy the application.

    ```
    ibmcloud app push
    ```

8. The route for your web application will be printed on the screen. Copy it to connect to it after the application successfully deploys.



```
Using manifest file [redacted]/manifest.yml
Getting app info...
Updating app with these attributes...
  name:                ibmlearner_translation_app
  path:                [redacted]
  disk quota:          1G
  health check type:   port
  instances:           1
  memory:              64M
  stack:               cflinuxfs3
  routes:
    ibmlearnertranslationapp.eu-gb.mybluemix.net

Updating app ibmlearner_translation_app...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
 4.29 KiB / 4.29 KiB [==========================================================================================] 100.00% 1s
```

> Troubleshoot: You can check the logs by running, `ibmcloud cf logs <appname> --recent`

9. You can see your deployed application listed in the Cloud Foundry. To make changes and push your app again, stop and delete the application from Cloud Foundry and then execute `ibmcloud app push`.

Congratulations!

You have completed the tasks for this project. In the peer assignement that follows, you will be required to upload the screenshots you saved in this lab.

Authors

Lavanya

Other Contributors

Ramesh Sannareddy

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-14	1.0	Lavanya	Created initial version of the lab