Name: Mr.Moghal Yaseen Pasha
Designation : Assistant Professor
Department : CSE-AIML
College name: Swami Vivekananda Institute of Technology(SVIT)
Mail id: yaseenpasham@svit.ac.in
Mobile Number: 9866504097

Project Title:Smart Assignment Engine

Category: **Cloud Application Development**

Skills Required:
Flask Integration, HTML, CSS, Bootstrap ,Docker , Kubernetes ,IBM DB2

Project Description:

Smart Assignment is a web-based application designed to streamline the process of assignment management and enhance collaboration among students, teach1ers, and administrators. This project aims to leverage web development technologies to create an intelligent platform that automates assignment creation, submission, grading, and feedback, thereby improving the efficiency and effectiveness of the assignment process. Students can access the platform to view and submit assignments. The application allows for secure file uploads, ensuring the integrity and confidentiality of submitted work. The system provides notifications and reminders to students about upcoming assignment due dates and any updates or clarifications from teachers.

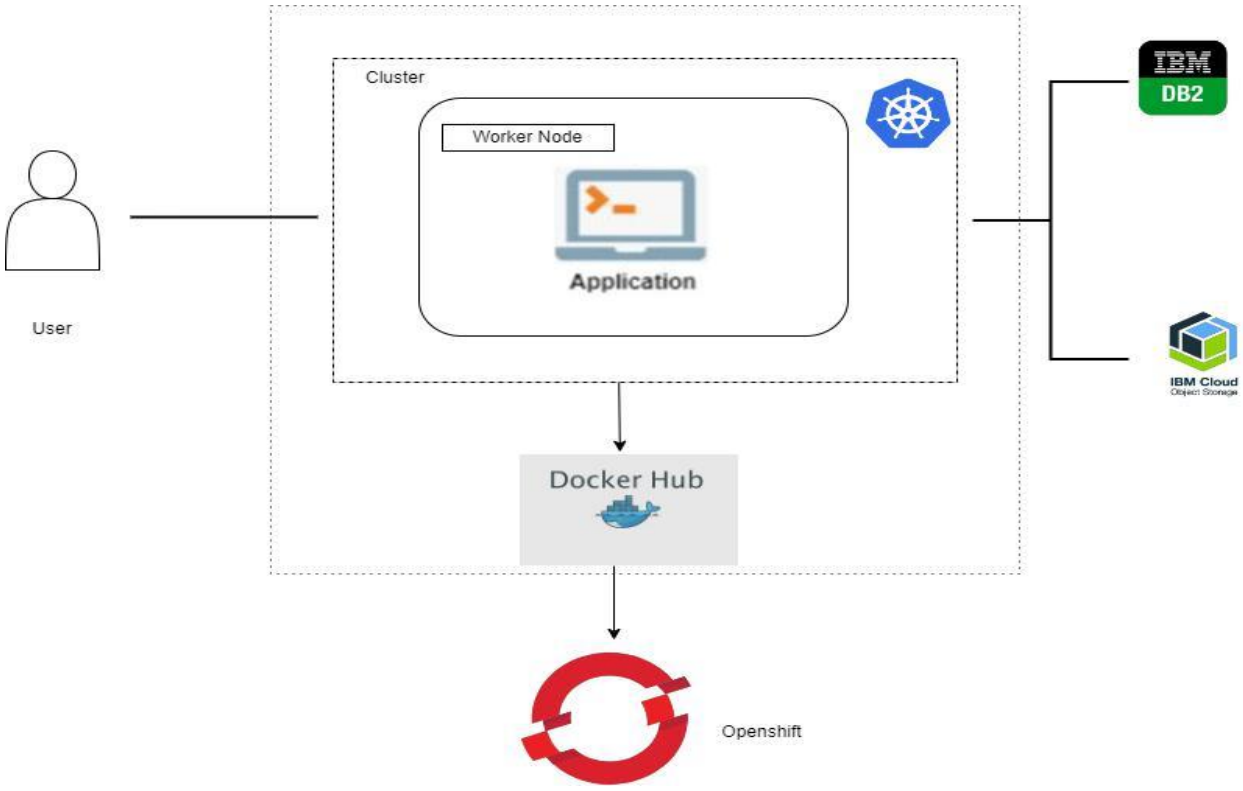
Smart Assignment aims to enhance the assignment management process, promote collaboration, and improve educational outcomes. By leveraging intelligent technologies and providing a user-friendly interface, the application streamlines administrative tasks, enhances student-teacher communication, and facilitates a more efficient and engaging learning experience.

Mentorship Program - Smart Assignment Engine

Smart Assignment is a web-based application designed to streamline the process of assignment management and enhance collaboration among students, teachers, and administrators. This project aims to leverage web development technologies to create an intelligent platform that automates assignment creation, submission, grading, and feedback, thereby improving the efficiency and effectiveness of the assignment process. Students can access the platform to view and submit assignments. The application allows for secure file uploads, ensuring the integrity and confidentiality of submitted work. The system provides notifications and reminders to students about upcoming assignment due dates and any updates or clarifications from teachers.

Smart Assignment aims to enhance the assignment management process, promote collaboration, and improve educational outcomes. By leveraging intelligent technologies and providing a user-friendly interface, the application streamlines administrative tasks, enhances student-teacher communication, and facilitates a more efficient and engaging learning experience.

Technical Architecture:



Project Flow:

To accomplish this, we have to complete all the activities listed below,

- **Define Problem / Problem Understanding** ○
  Specify the business problem
  ○ Business requirements
  ○ Literature Survey
  ○ Social or Business Impact.

- **Creating a User Interface**
  - Create a User Interface using HTML, CSS/Bootstrap, and JavaScript, where the user will interact and give input.

- **Database Connection**
  - Creating necessary Schemas and Tables
  - Create a Cloud Object Storage to store multimedia data.

- **Flask Application**
  - Creating python's flask web framework to connect the front end with the back end services.

- **GitHub**
  - Pushing all the files to github by using git commands

- **Containerization**
  - Containerize your application as a docker image and push it into Docker
  - Push the image to Docker Hub.

- **Kubernetes**
- Creating a YAML configuration file

- **Deployment Application in Openshift**
  - Deploying with Git Repository
  - Deploying with Docker Image
  - Deploying with YAML File

Define Problem / Problem Understanding

Specify The Business Problem

Refer to Project Description

Business Requirements

Business requirements of a smart assessment engine encompass several key aspects. Firstly, it should possess robust functionality to create, deliver, and evaluate assessments efficiently. This includes the ability to generate various question formats, support adaptive testing, and provide instant scoring and feedback to users. Secondly, it should offer seamless integration with existing learning management systems or platforms, enabling easy administration and data management. Additionally, the smart assessment engine should prioritise data security and privacy, ensuring the confidentiality and protection of sensitive user information. It should also have scalability to handle a large number of concurrent users and accommodate future growth. Furthermore, the engine should be adaptable and customizable to meet the unique needs of different educational institutions or organisations, allowing for personalised branding and tailored assessment experiences. Lastly, it should offer comprehensive analytics and reporting capabilities, enabling administrators to gain insights into performance trends, identify knowledge gaps, and make data-driven decisions to improve the assessment process and overall learning outcomes.

Literature Survey

A literature survey of smart assessment engines reveals a significant body of research focusing on the development and implementation of intelligent systems designed to revolutionise the assessment process. Studies demonstrate the potential of smart assessment engines to improve efficiency, accuracy, and fairness in evaluating student performance while providing personalised feedback and adaptive learning experiences. Furthermore, research highlights the importance of designing user-friendly interfaces, ensuring the privacy and security of data, and addressing potential biases and ethical concerns associated with these systems. Overall, the literature showcases the promising role of smart assessment engines in transforming educational assessment practices and advancing the field of assessment technology.

Social Or Business Impact.

Social Impact: It will make the life of a student on campus much easier. Professors also no need to carry the big pile of notebooks submitted by students and the transparency of marks allotment also will be there.

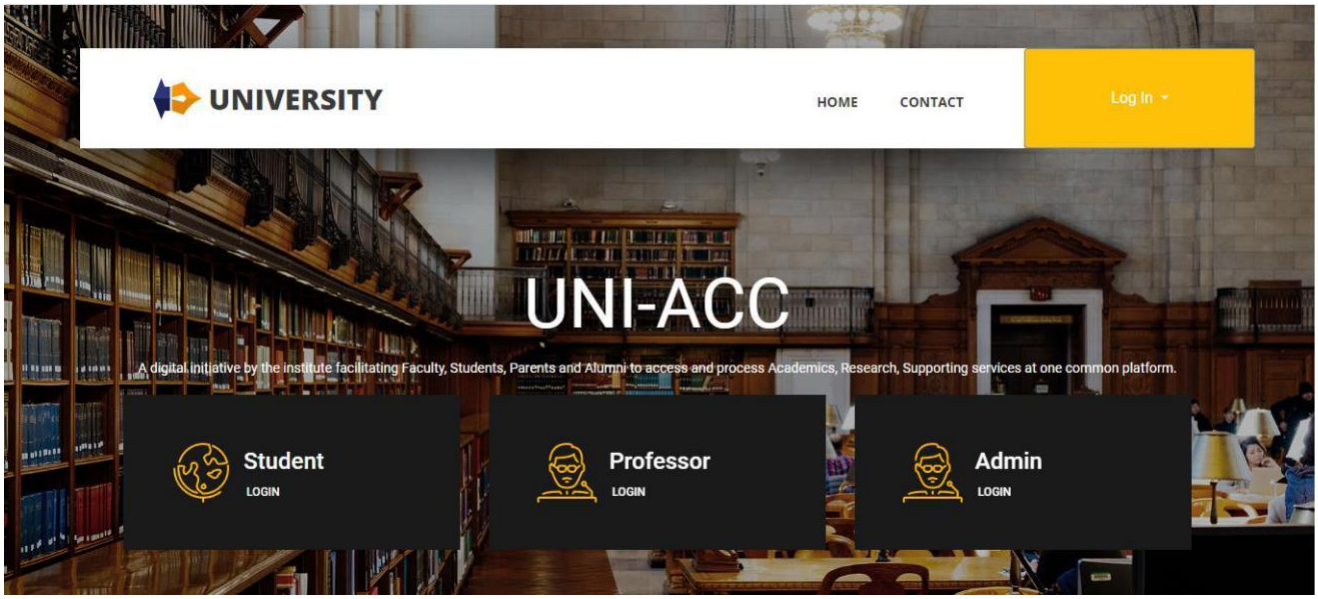
Business Model/Impact: It will also give a positive impression about the university around the time of admissions and can be made by digitizing the campus.

2. Creating A User Interface

User interfaces (UI) provide a way for users to interact with a system or application. They are designed to make it easy for users to access and control the functionality of a system. User interfaces play a critical role in the overall user experience by providing a way for users to interact with a system in a simple, efficient and effective manner.

2.1 Creating HTML Pages

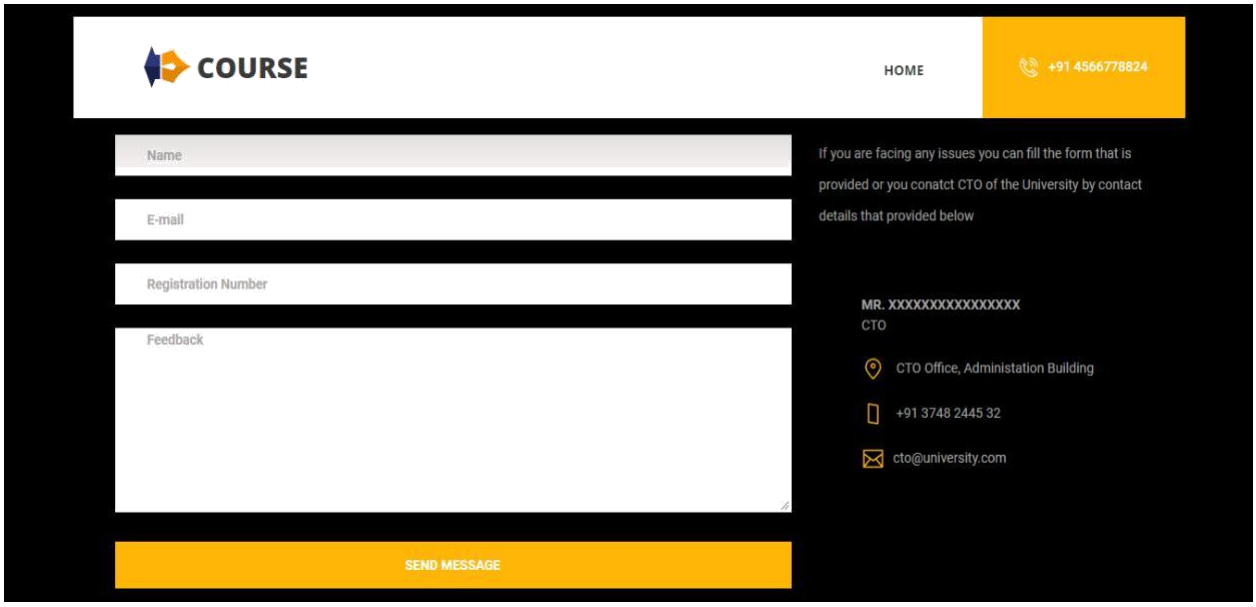
Activity 1.1: index.html
Index.html is a homepage, this html page will have it as the landing page of our college website.



As we can see in the above picture of index.html where we have login options for students, professors and Admin (who manages the data).

If we click on the contact button it will take us to the contact page that we code as contact.html

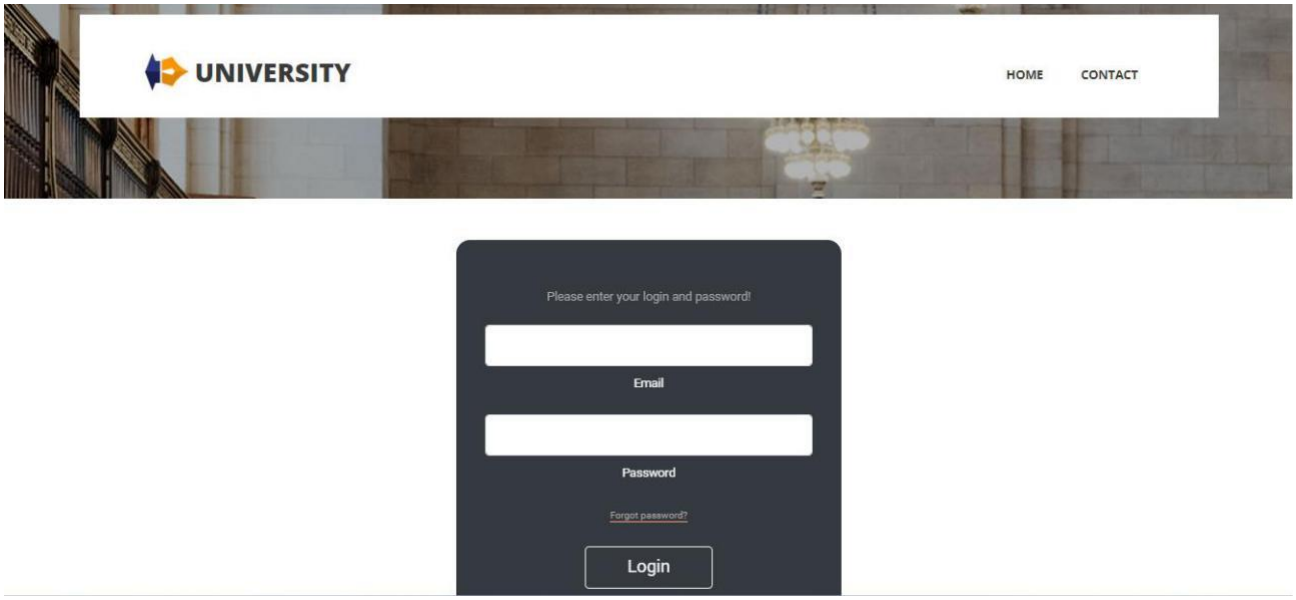
Activity 1.2: contact.html
In this html page, we are going to give a form where users can give feedback for any issues they are facing or can also contact the concerned person by using the contact information provided.



If we want we can save this feedback in the database and get it to the admin page so that admin can see and respond to the issue.

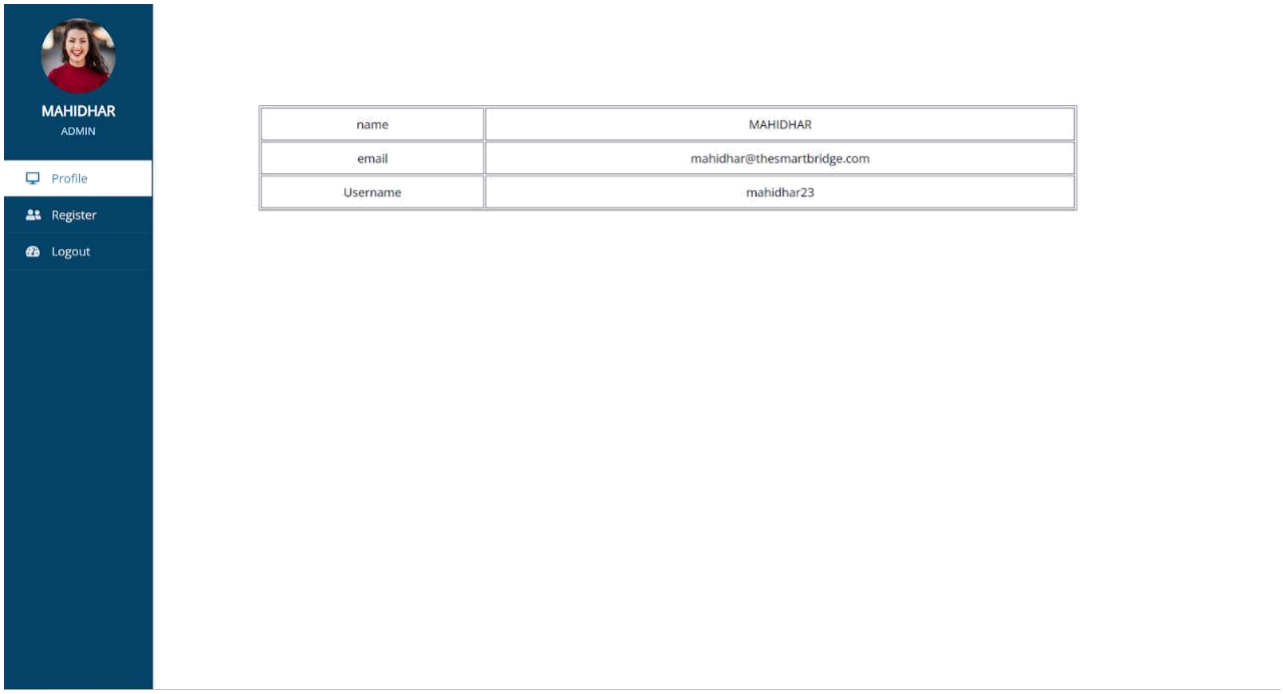
Activity 1.3: login.html

In this html page, all the people will log in. Here we don't have a register section because registration will be done only by the admin in their login. When an admin registers a mail will be sent. Here they will use that login credentials.



Activity 1.4: adminprofile.html

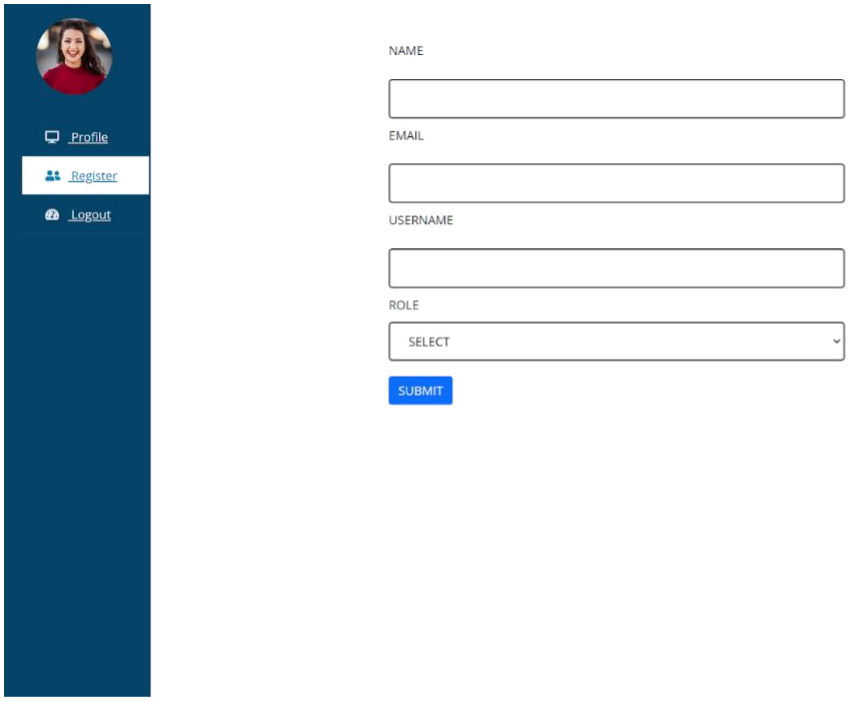
In this html page, when the admin login with their credentials with the backend code of routing we will find whether it's admin or not. If yes then we will route him from login to this html page.



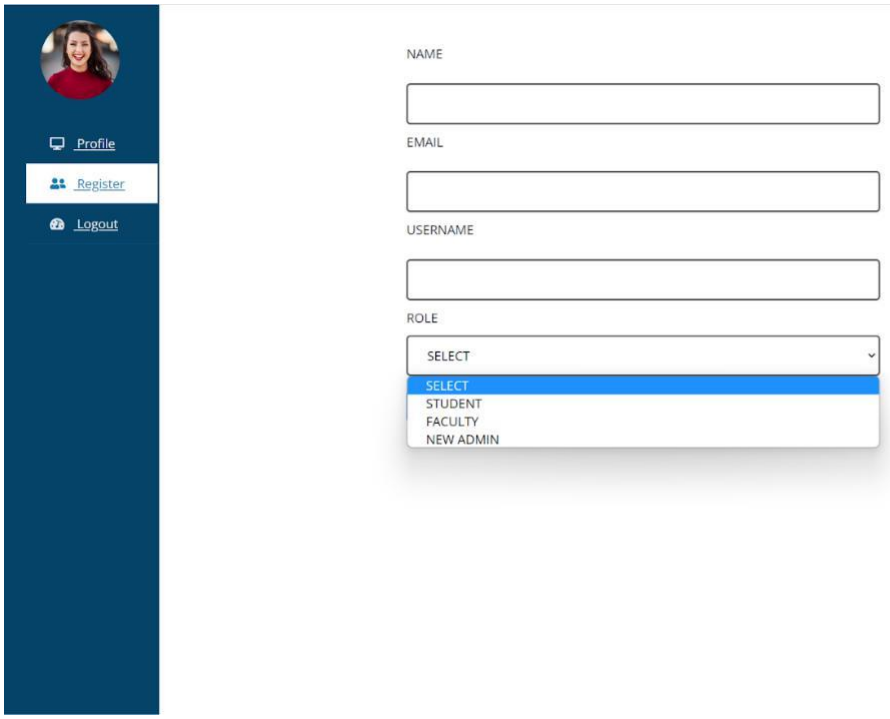
Once they successfully log in as admin, adminprofile.html is the page they will be redirected to where they can see their details like photo (which is on the left side top) username, email and name. We can also display more info like phone number, etc…

Activity 1.5: adminregister.html

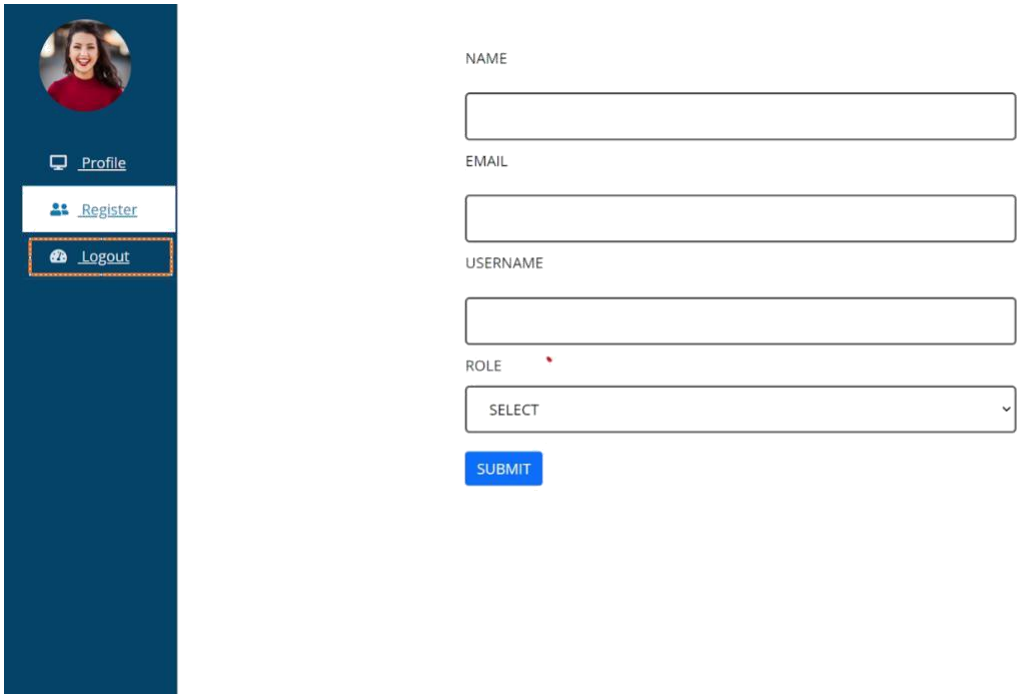
Now, when the admin clicks on that register button which is on the left side(highlighted with an orange colour box), they will be redirected to the registration page where they can register the students, professors and other admins too.



When we click on the select of role, we will get a dropdown to add people.



When you click on, the logout button on the left side (as shown below picture). The user will be redirected to the index.html page (common for all users).



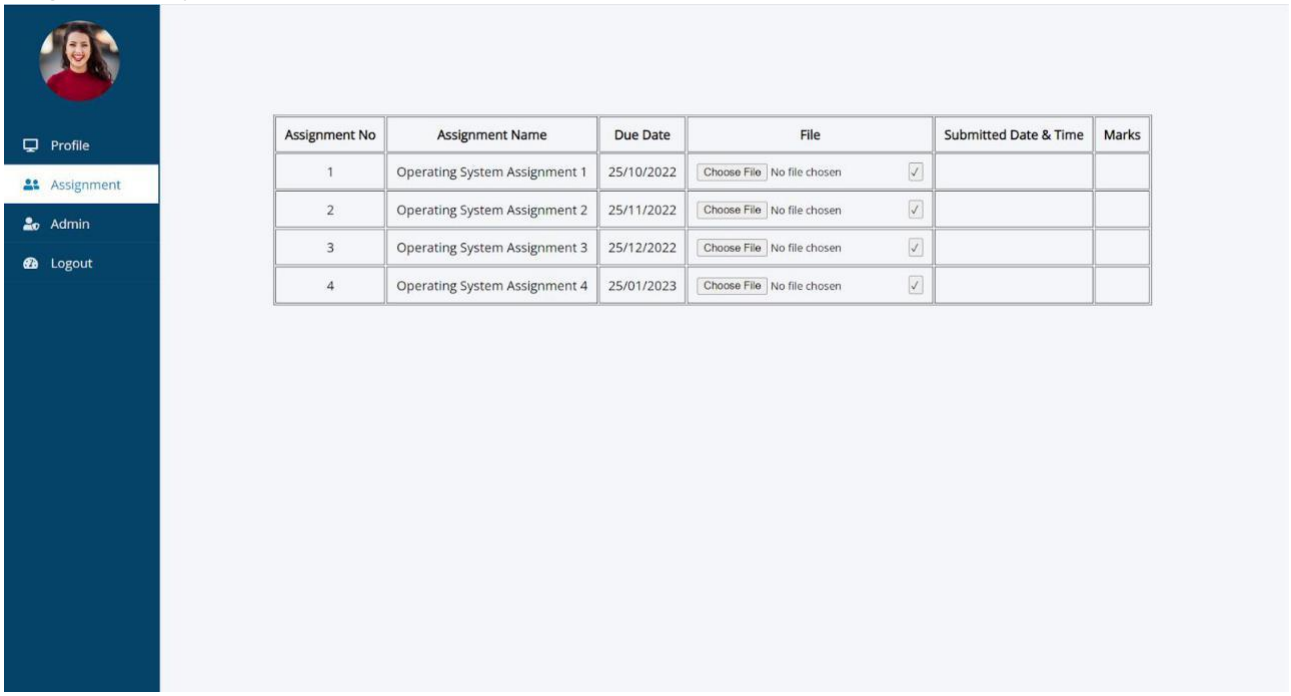
Activity 1.6: studentprofile.html

This html page is also the same as adminprofile.html. Where students can see all the details of the student along with a profile picture, here also we can add more like the year, registration number, branch, etc…

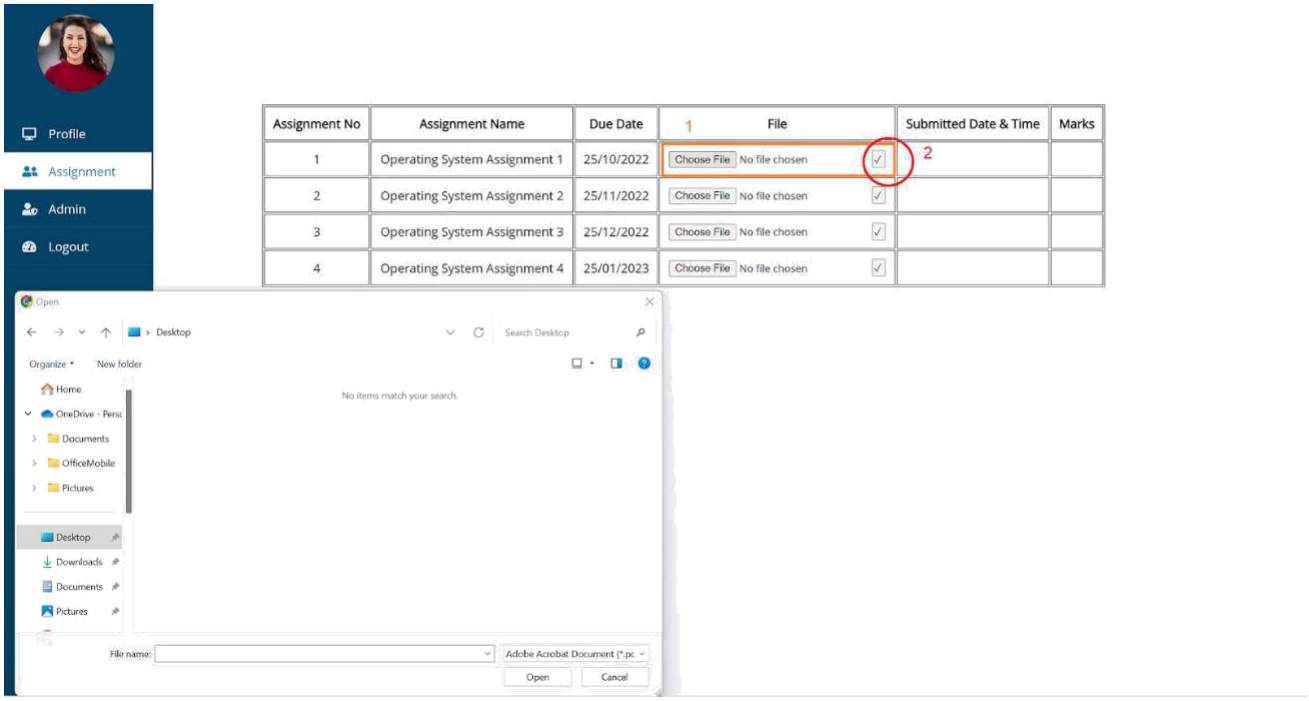
Activity 1.7: studentsubmit.html

When students click on the assignment button they can see a list of assignments that they have to submit.

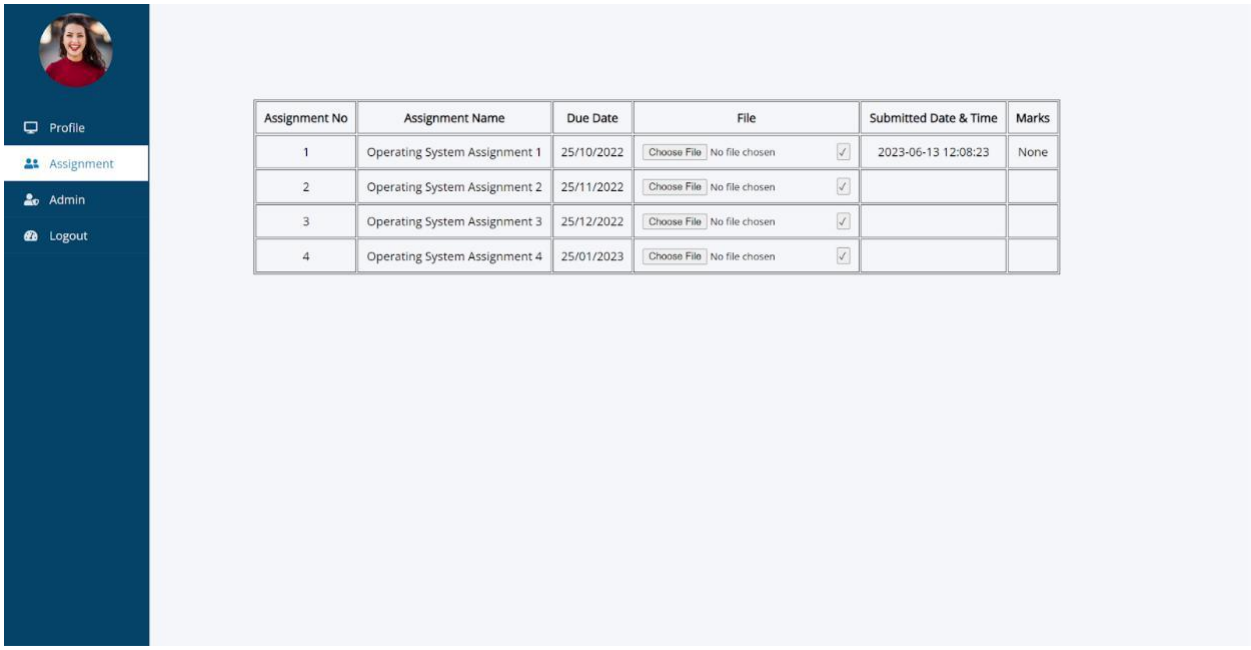
As you can see, here is the list of some assignments where students can know what and how many assignments they have to do and the last date of submission.



Students can click on the choose file button(highlighted in the orange color box), Here we have asked them to upload only pdf files.



Once they selected and submitted the file by clicking on that tick button (2), once they submitted the file. The will be submitted into object storage in the backend and the submit time will be shown for the students next to the submit button.



Note: Here we can also make a drop down where students can select subjects and based on the subject students can see the assignments of the particular subject.

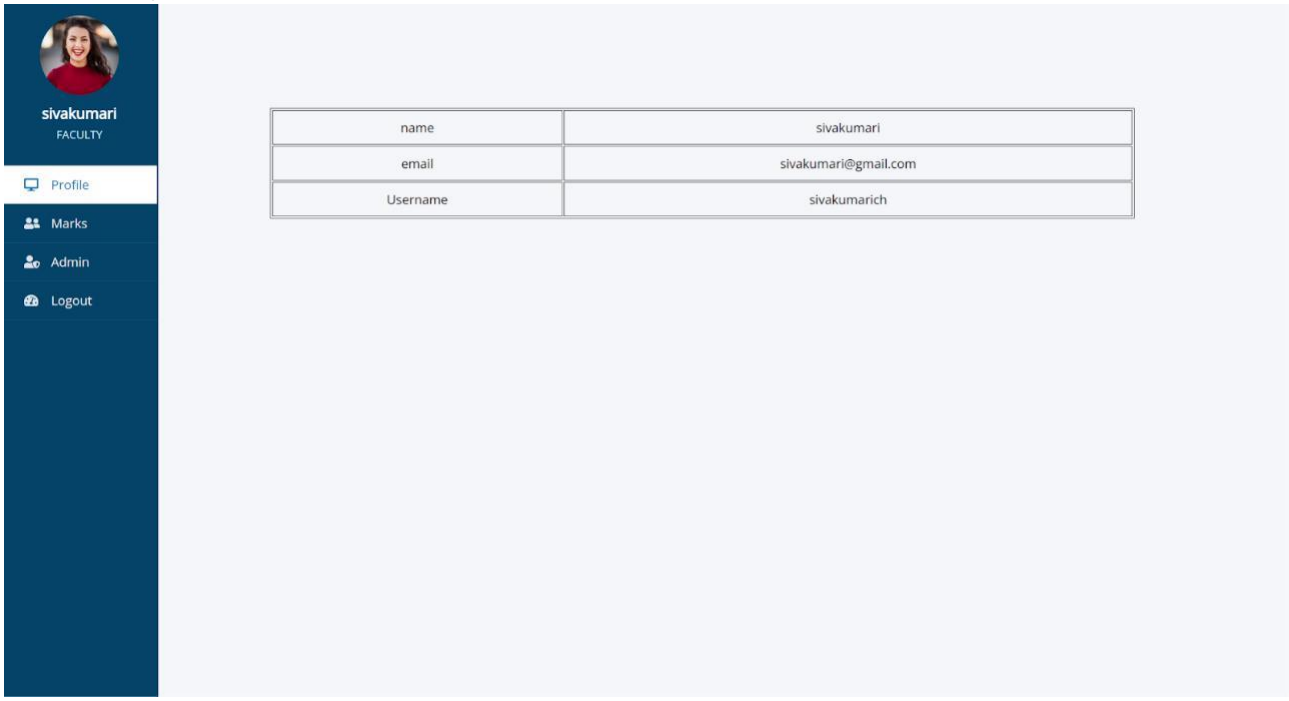
Note: Marks ? None — means not yet assigned.

Admin - Contact Admin so, it takes us to the contact.html page

Logout - Same as it does in the admin page too. It will log out the user and redirect to index.html

Activity 1.8: facultyprofile.html

In this html page, same as other profiles pages. We will have all the information about the faculty. Here instead of making three different html pages we can make it into a single page but need to figure the info we have shown, then we can make a dynamic page regarding it. To make it more dynamic it is a little bit tricky.



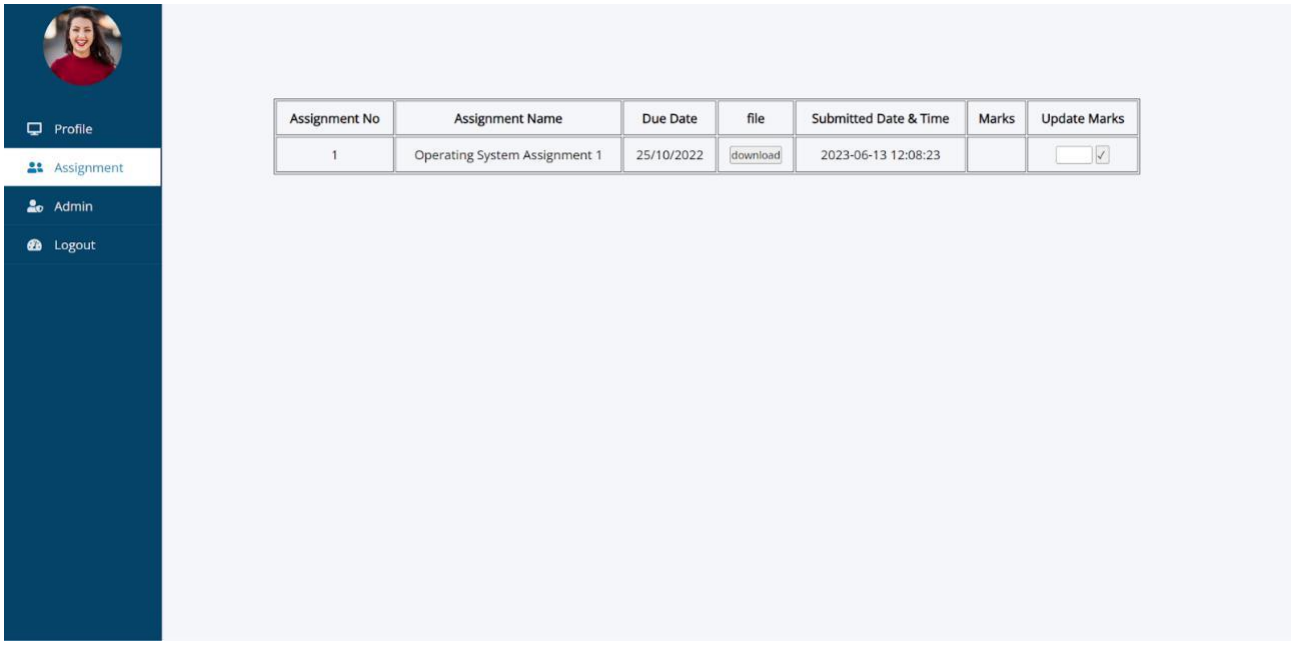
Activity 1.9: facultystulist.html

When you click on the marks button you will be redirected to this html page. In this, we can see all the lists of students that are enrolled for this course you are teaching.

Now to see the submissions they have done, you click on the submissions button (highlighted in the orange color box in below picture)

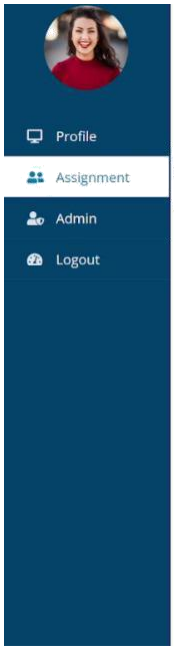
When you click on the submissions button you will be redirected to the facultymarks.html

Activity 1.10: facultymarks.html
In this html page, we will see the submissions made by the student and also we can give the marks.



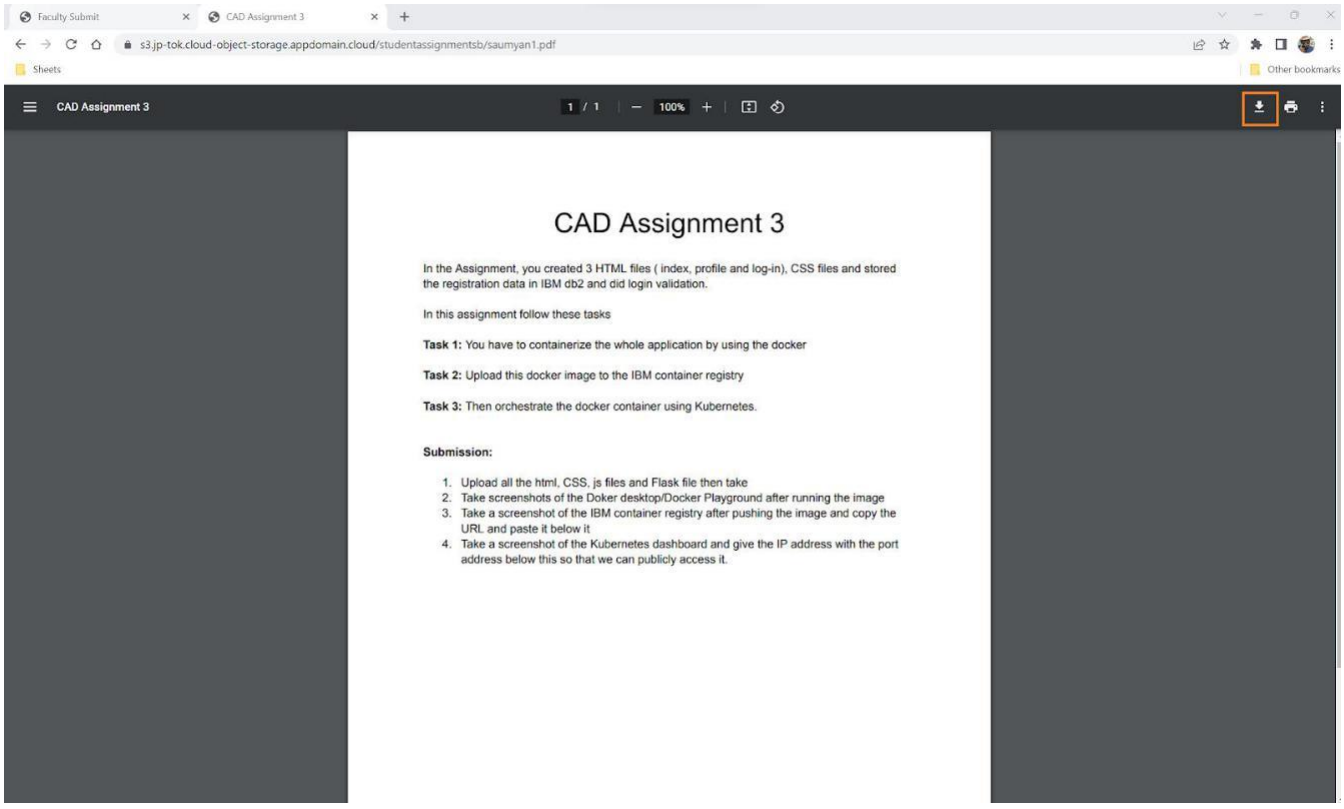
Here you can see the table, but only with 1 submission, because previously just above we have submitted only one file.

Faculties can click on the download button, it will open the file in a new tab. Where you can see the file or you can download the file to your computer.

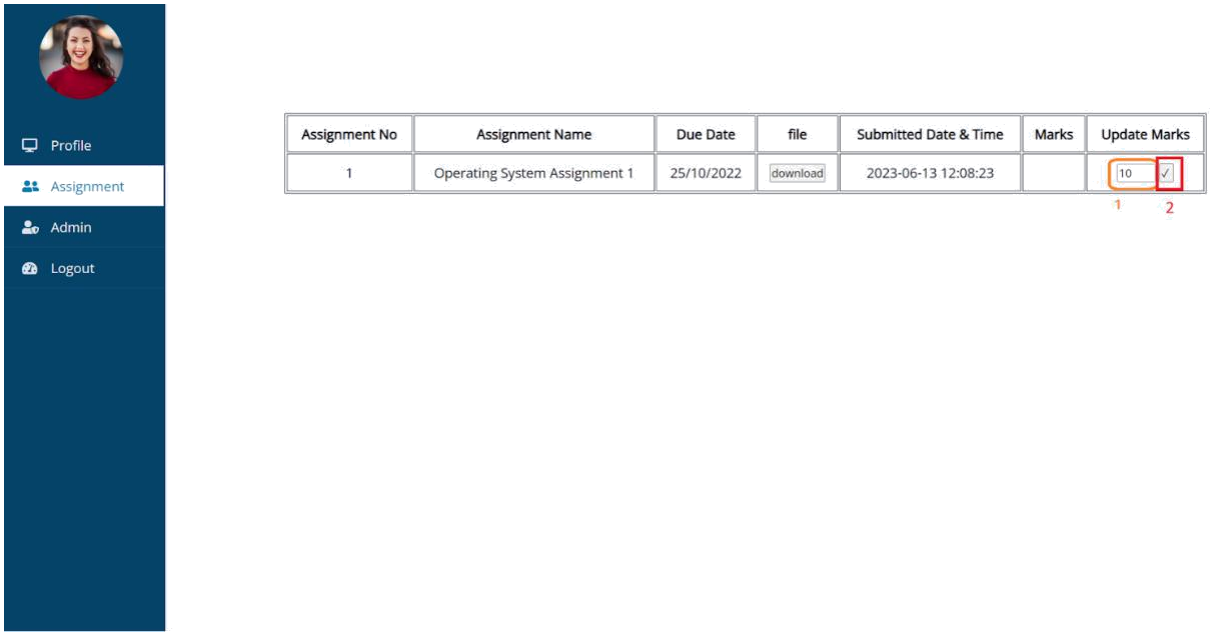


Assignment No	Assignment Name	Due Date	file	Submitted Date & Time	Marks	Update Marks
1	Operating System Assignment 1	25/10/2022	download	2023-06-13 12:08:23		

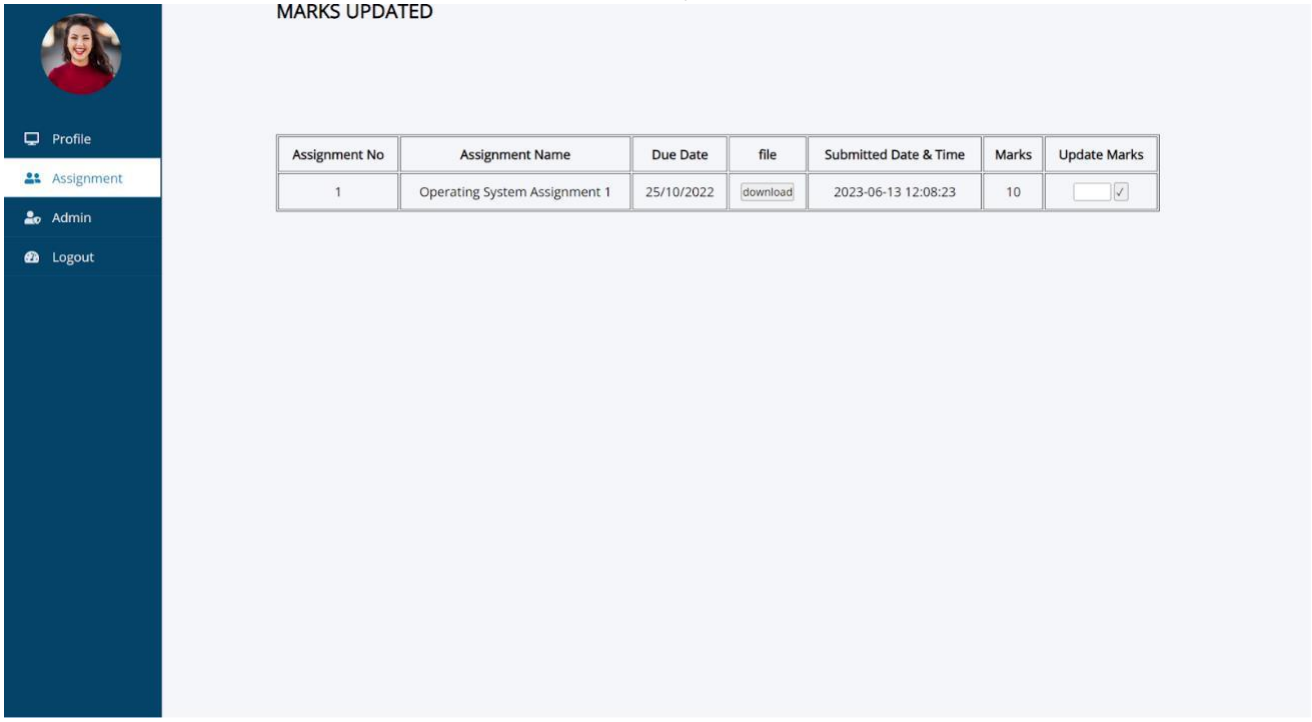
To download the file click, on that download button (highlighted in orange color)



Now to give marks after evaluating you can give in the box and then click on the tick button.



To give marks
To submit the marks given
Once you click on the update/ tick mark button it will be displayed in both the marks column of faculty and student, and will be stored in the database simultaneously.



Below are the marks updated in the student account.

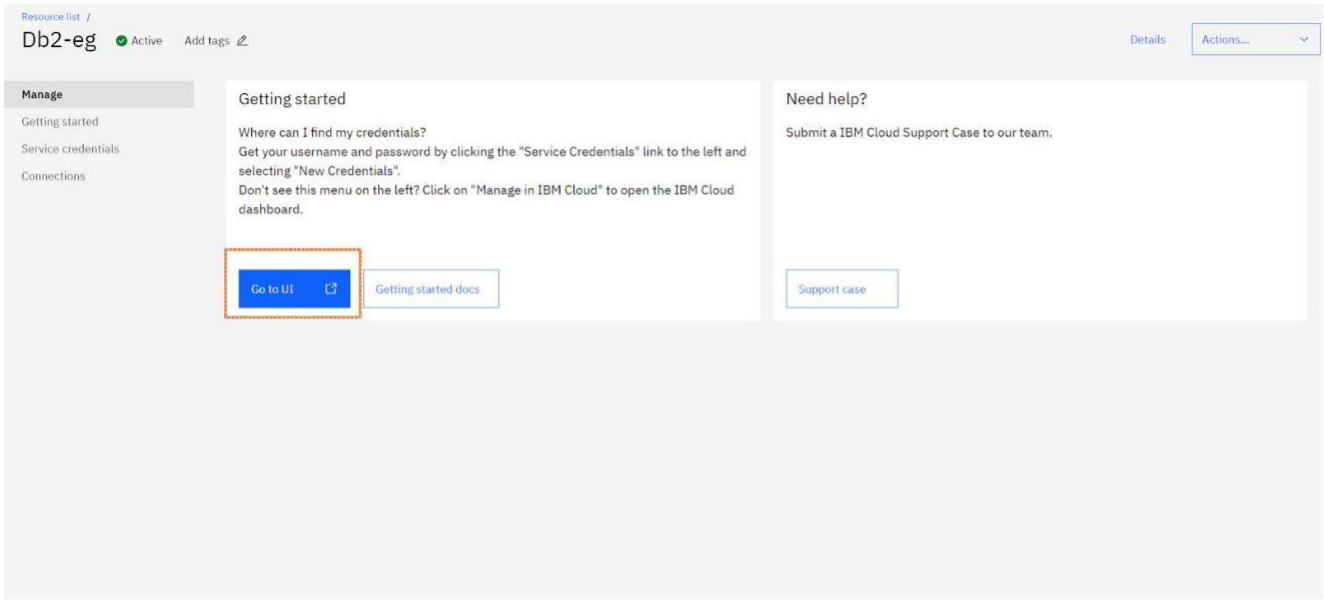
Assignment No	Assignment Name	Due Date	File	Submitted Date & Time	Marks
1	Operating System Assignment 1	25/10/2022	Choose File No file chosen	2023-06-13 12:08:23	10
2	Operating System Assignment 2	25/11/2022	Choose File No file chosen		
3	Operating System Assignment 3	25/12/2022	Choose File No file chosen		
4	Operating System Assignment 4	25/01/2023	Choose File No file chosen		

3.

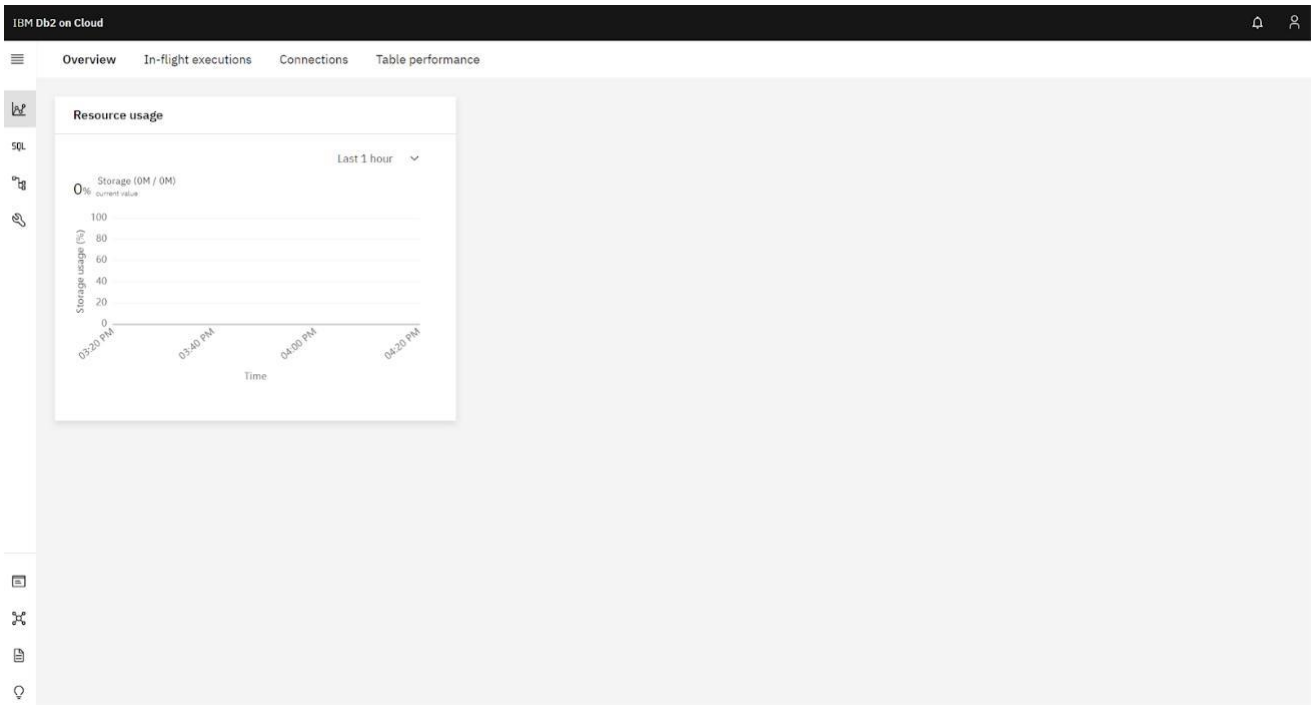
Database Connection

Activity 1.1: Creating Register Table

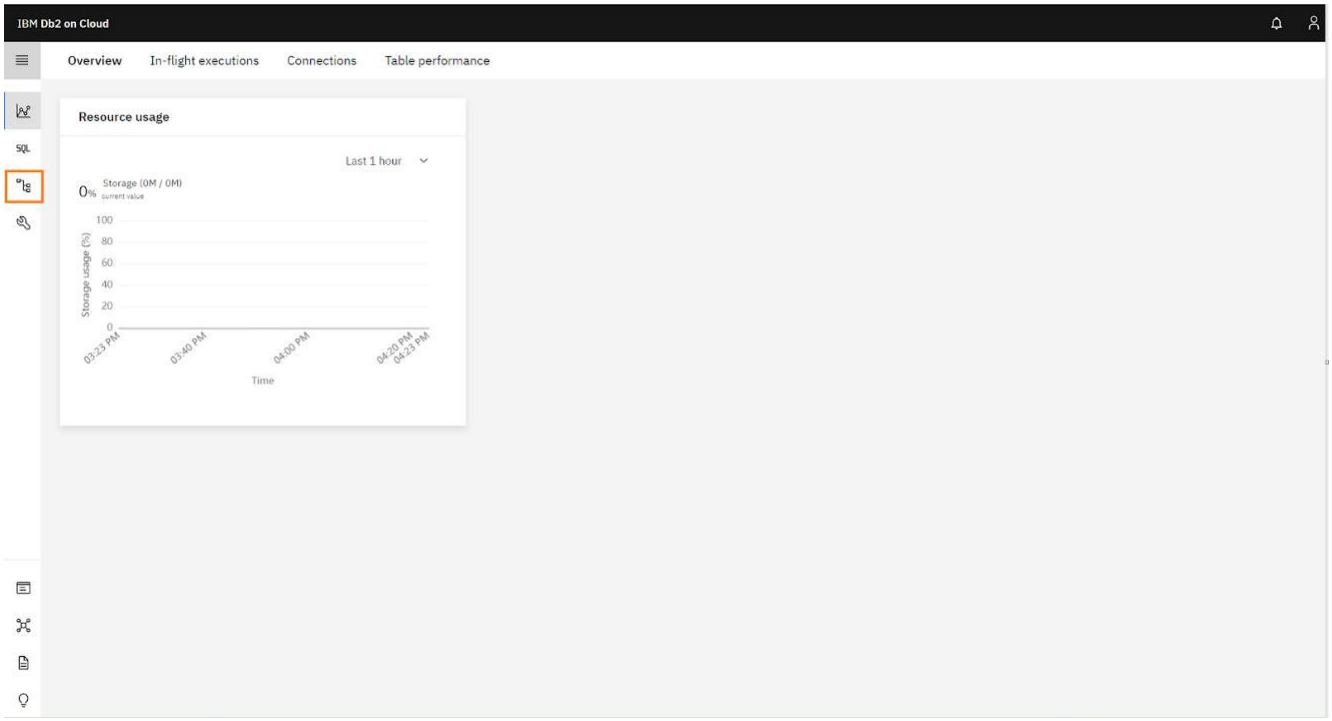
When you create DB2 service in IBM Cloud you will be redirected to this page



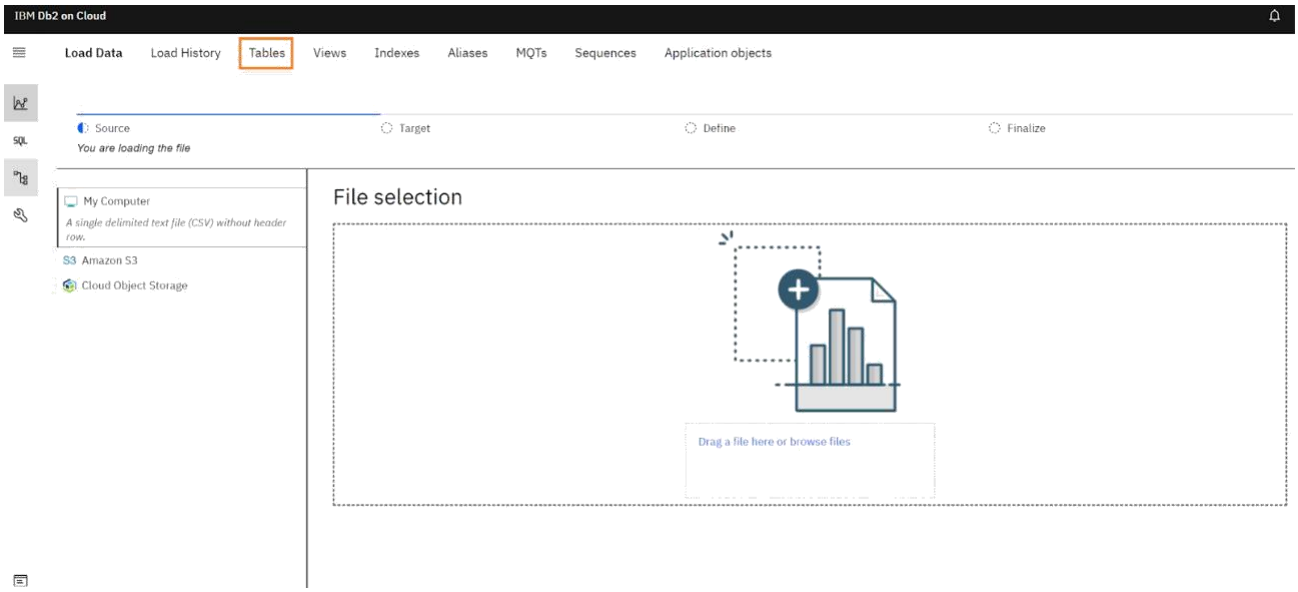
Once you click on the Go to UI button, it will take you to the db2 page.



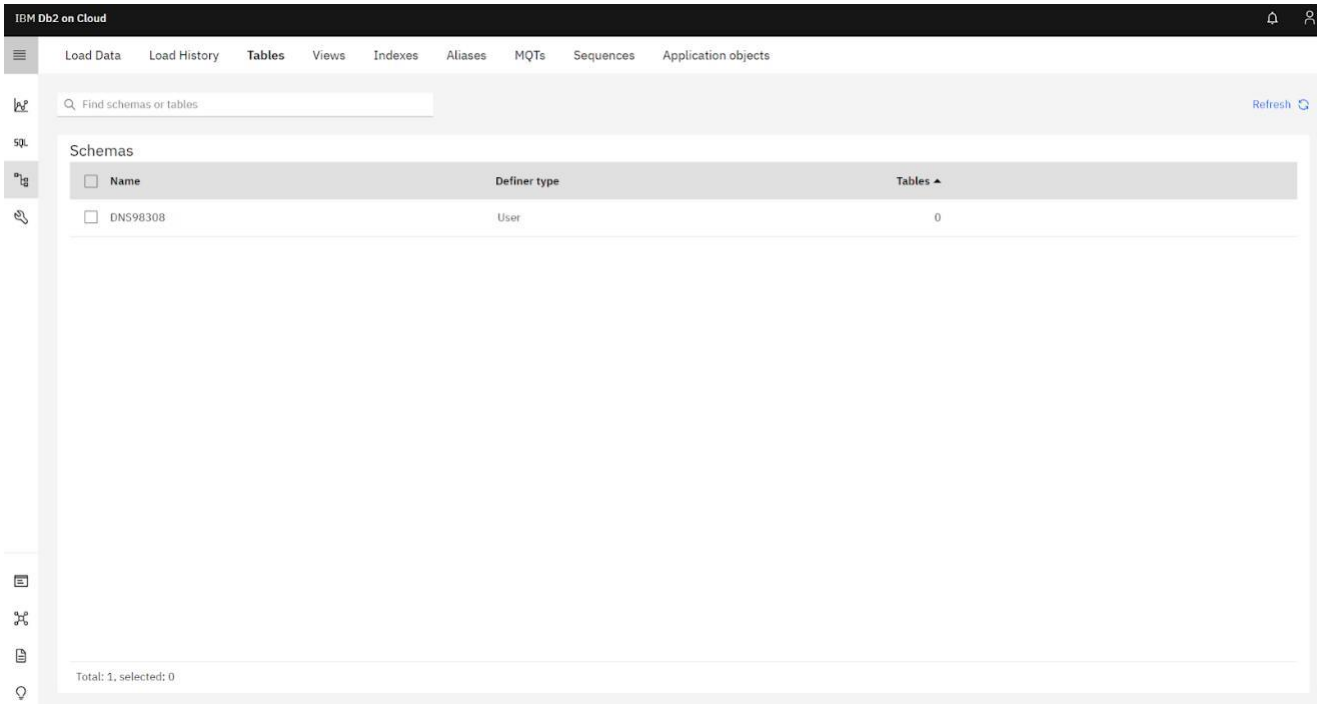
Now click on the data symbol (highlighted in the orange colour). This will take you to the page where you can see the data and its performance.



Here you can upload the data from computer, amazon s3 storage or cloud object storage. Now we are going to concentrate more on the tables tab on the top.

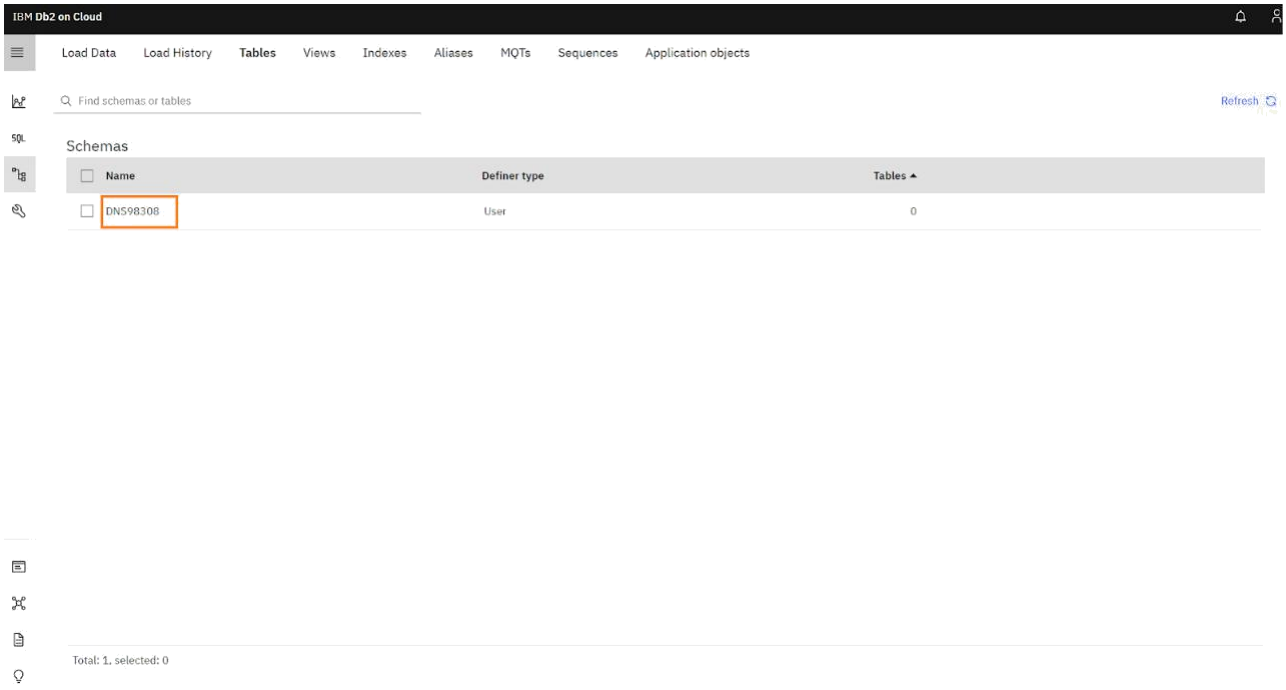


Click on it, you will be redirected to the schema page of ibm db2. Where you can see the schema name, no of tables you have in your schema.

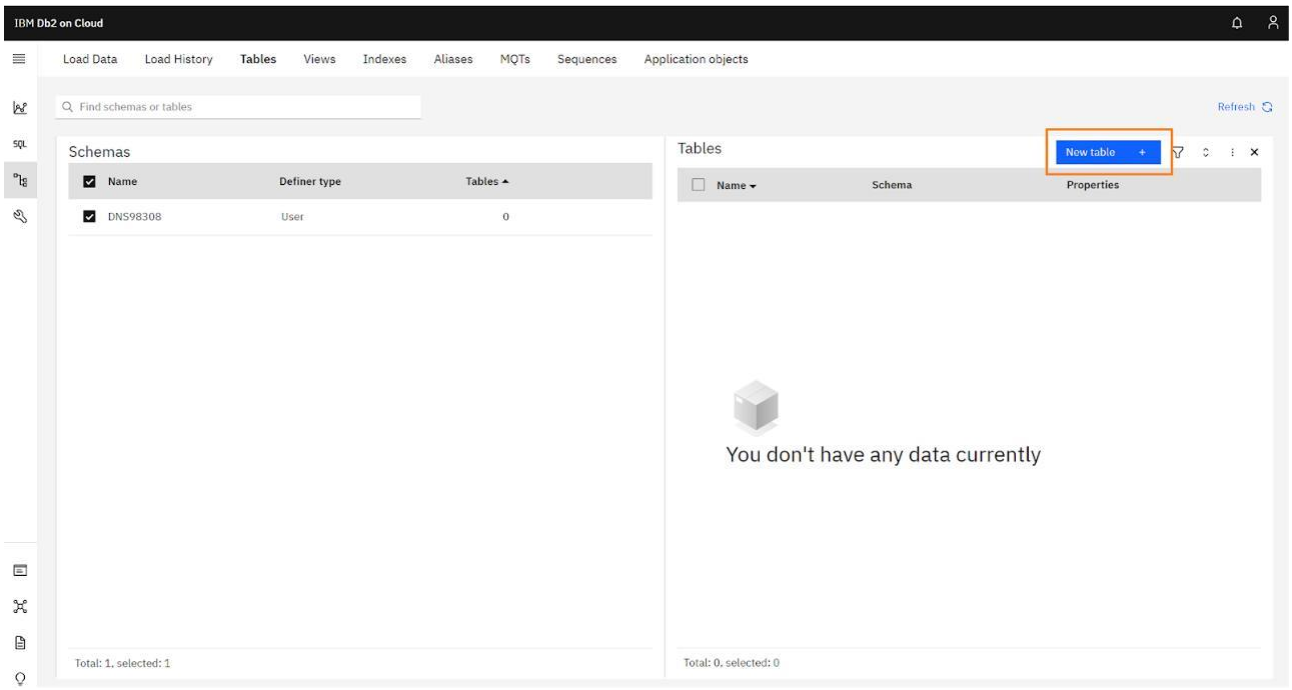


Note: If you are using trail accounts of ibm, you can only have 1 schema

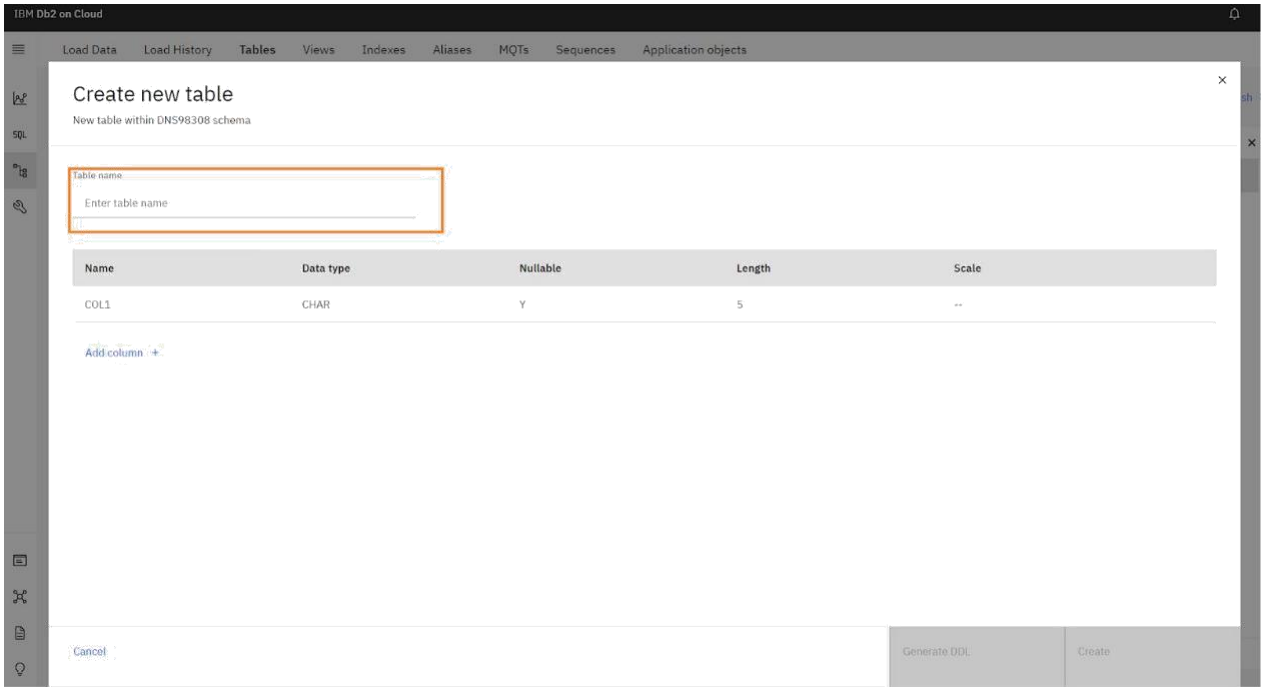
Now click on the schema name



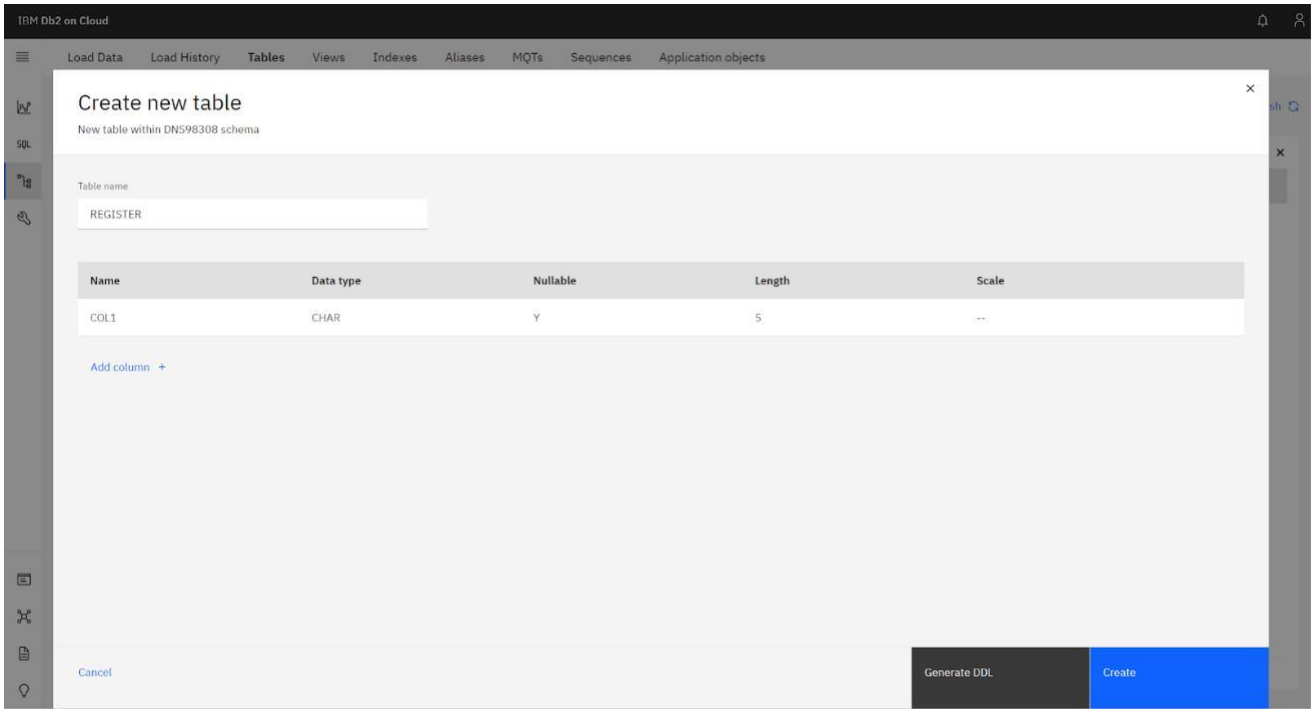
Then you see a new tab that opens on the right side, as of now that will be empty, from there we can create a table by clicking on the new table button which is in blue colour.



Now it will open a window, where you can give the name for your table.

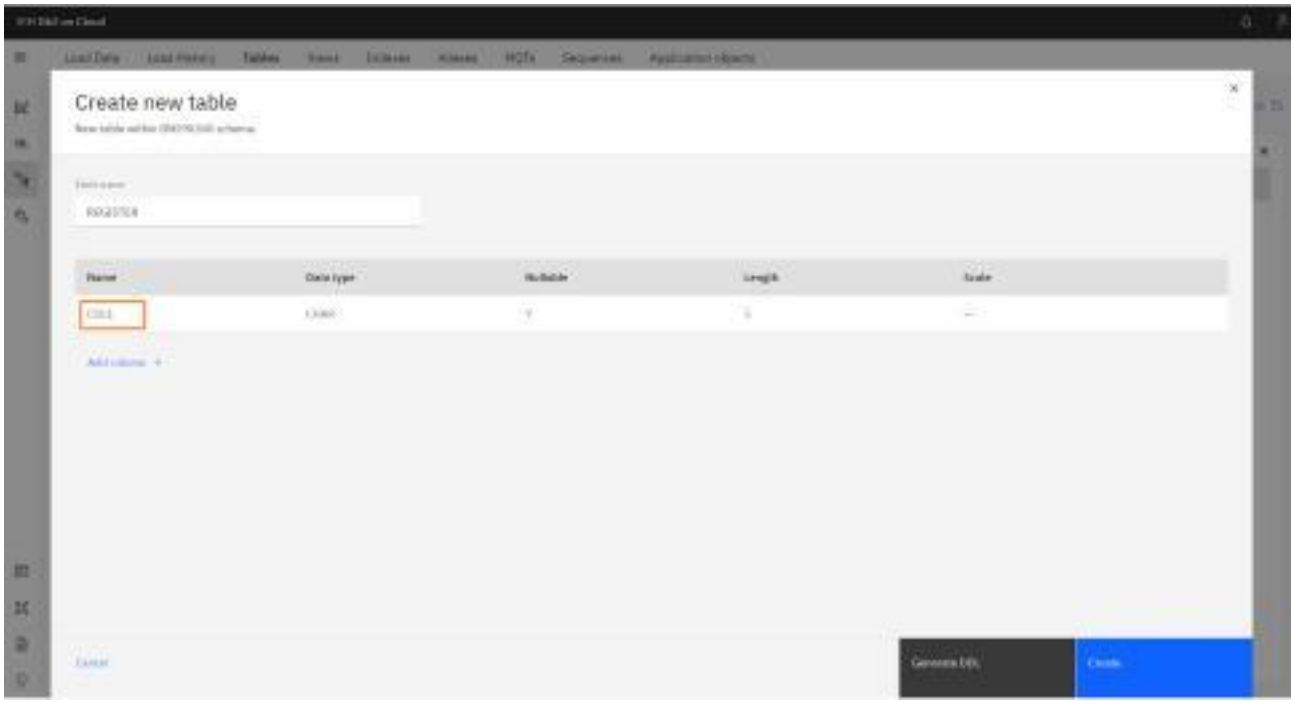


Click on the empty field box that is highlighted to give the name. First we can create a register table, that we will use to store the details of all people in the university and we will also use this table for login validation too.

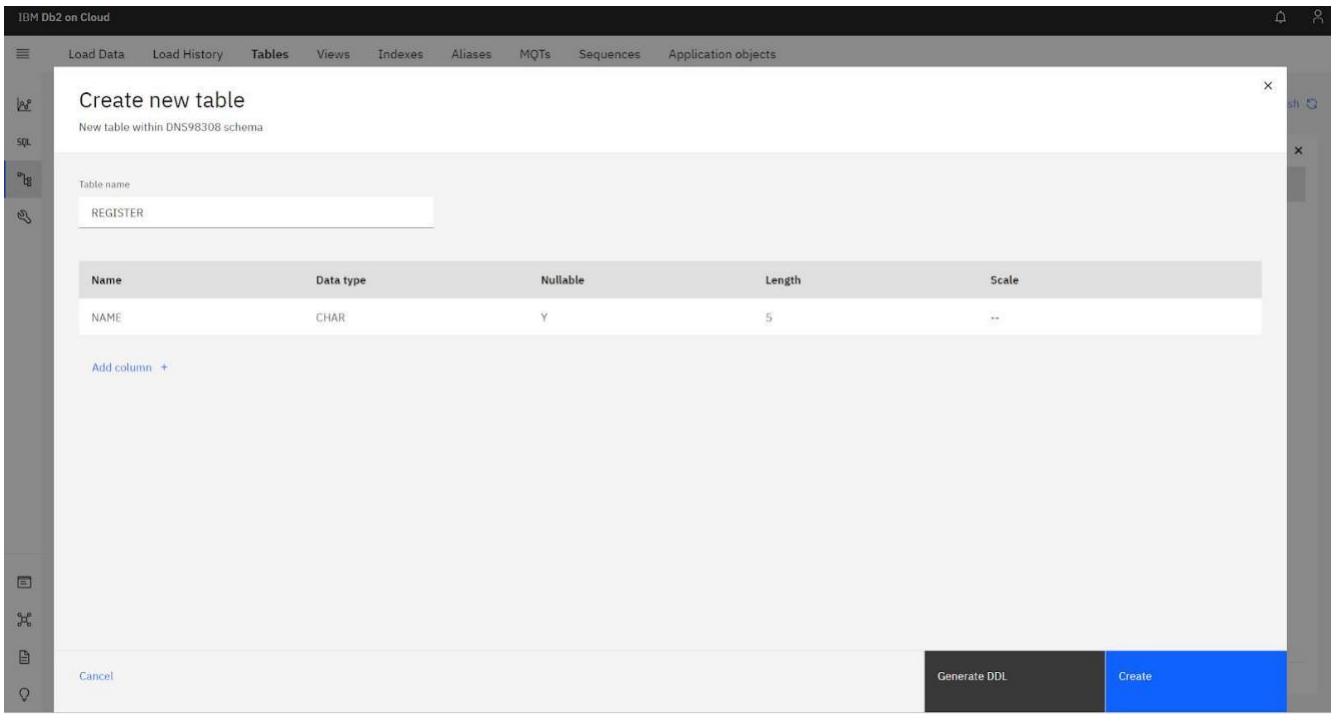


The columns that we can create is completely based on the information that we are taking or we want take from the html page
The information that admin going to give in this project while registration is NAME, EMAIL, USERNAME, ROLE (this is for to differentiate between faculty, student and admin, instead of having separate tables we can club this info by adding one more column), PASSWORD (this will be auto generated by a python code)

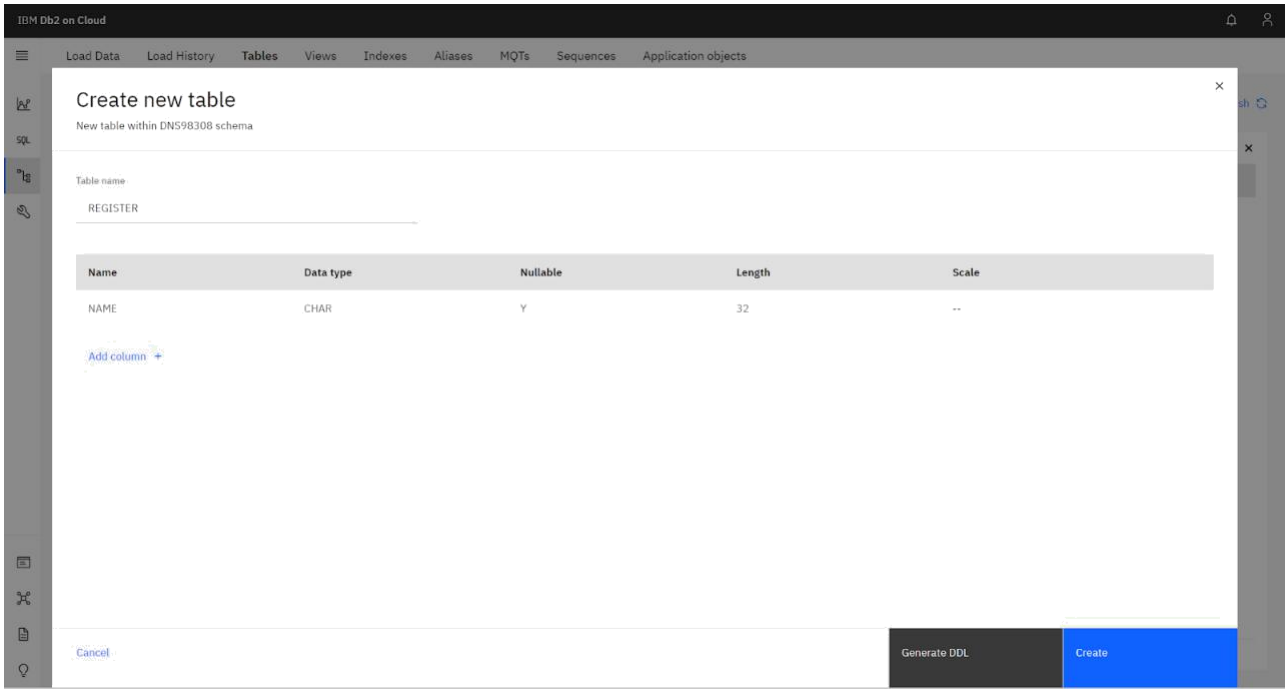
Now to give the 1st column name, we can click on that COL1 thing just under the Name heading.



When you hover on it, edit (pencil type thing) will popup. Click on it and give first column name i.e NAME

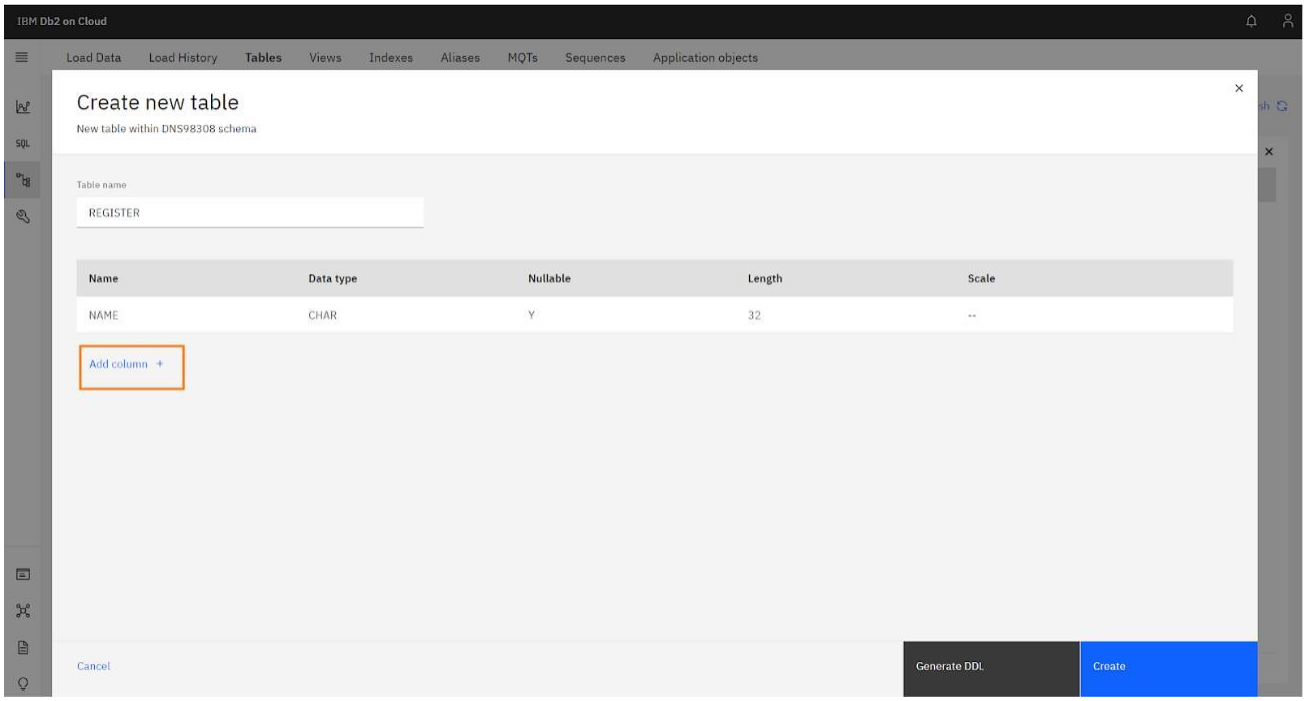


The next things like Data Type, Nullable, Length, Scale also we can change in the same way that we changed the Name heading.

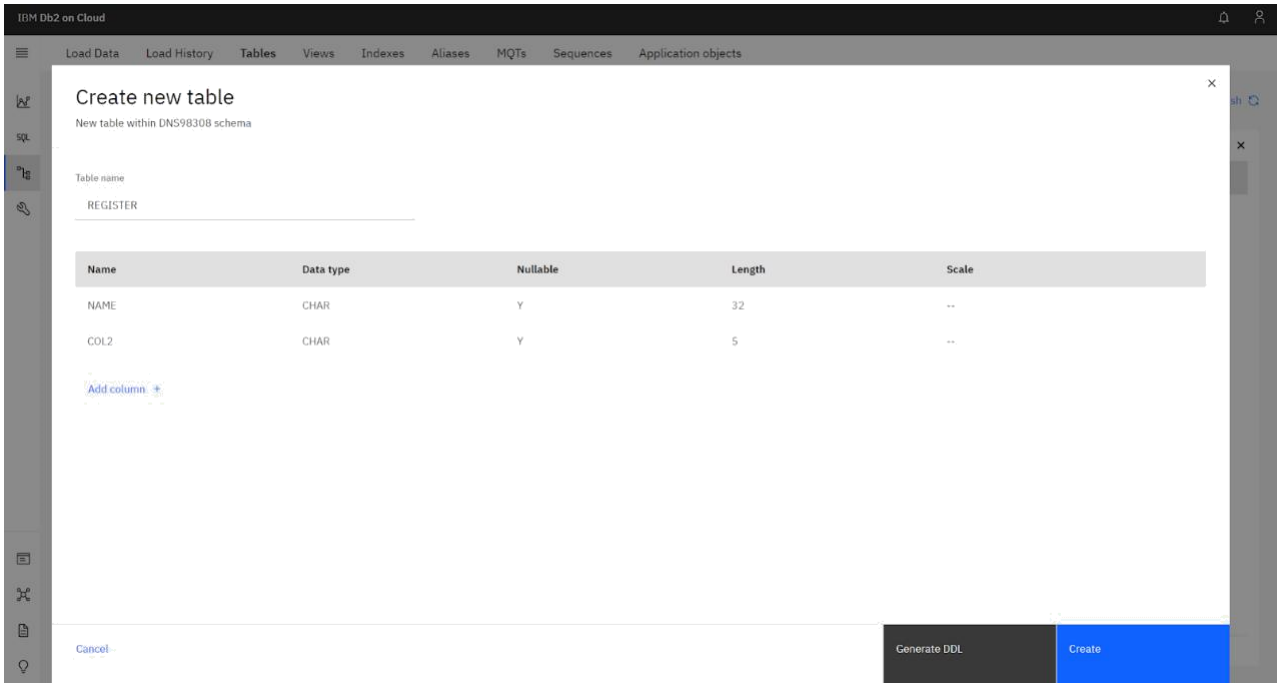


Here We can take length of name as 32 because some will have a big name (32 is nothing but max number of characters that it should store)

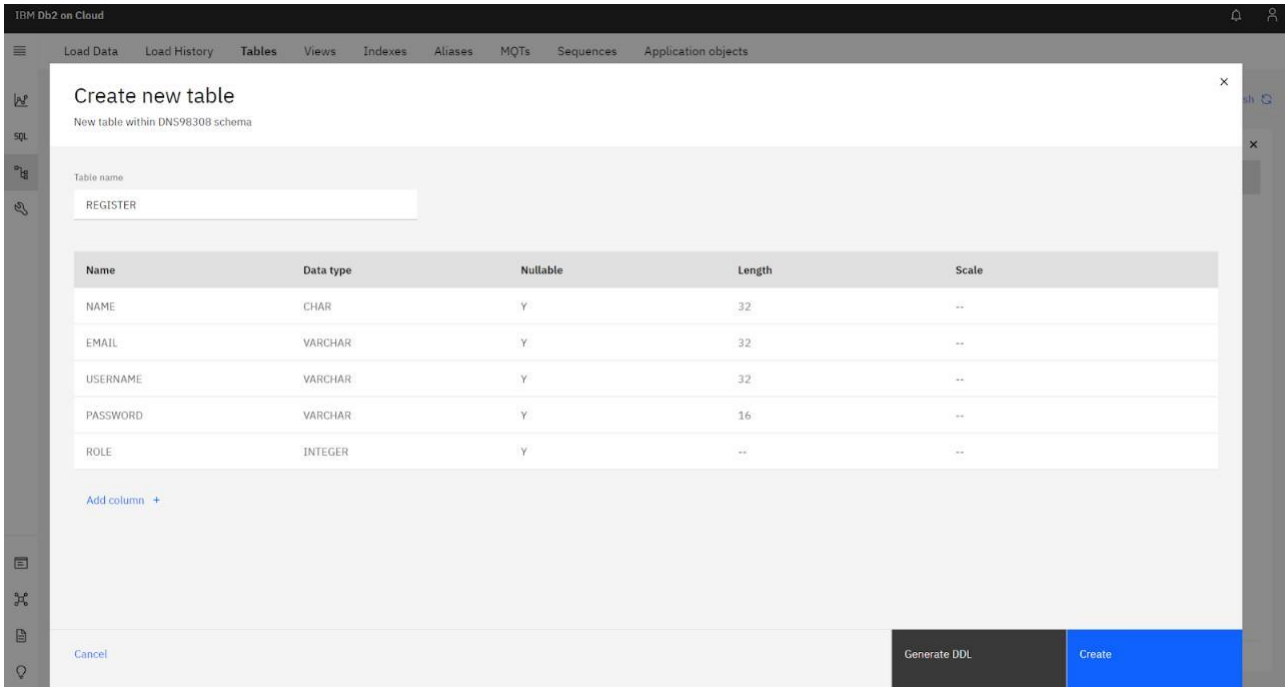
Now to add a next columns click on that Add Column + button



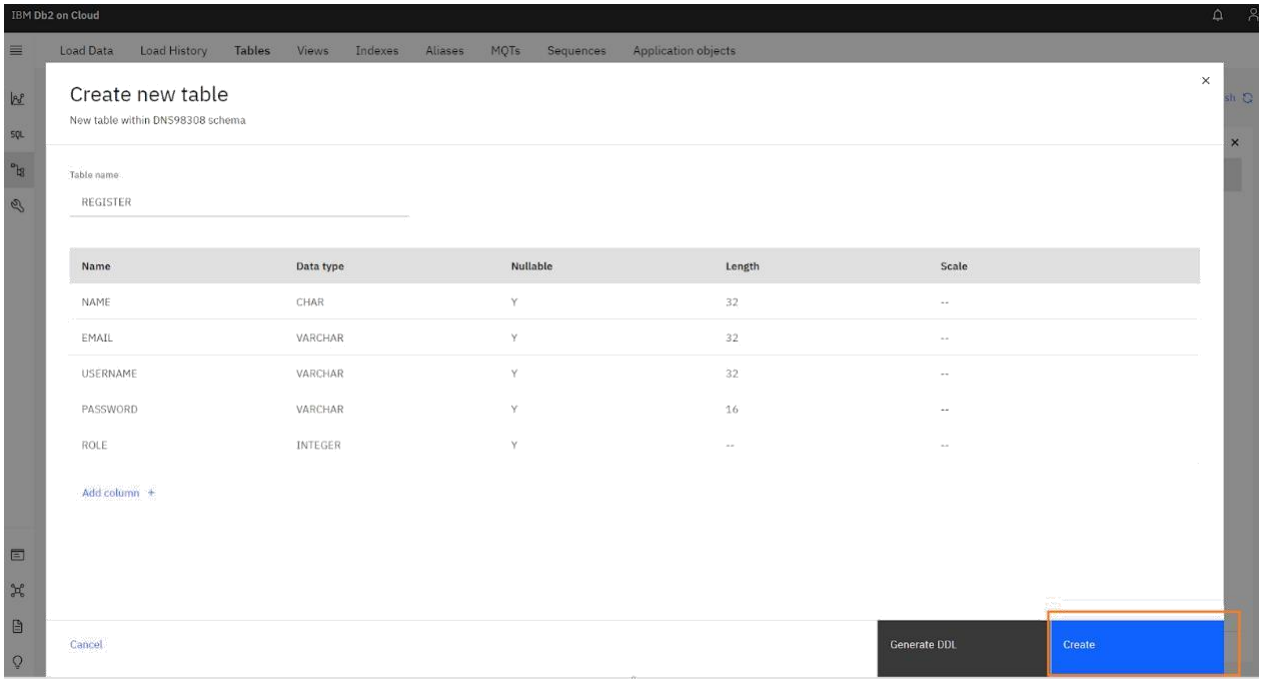
When you click on it, a new row add column will appear.



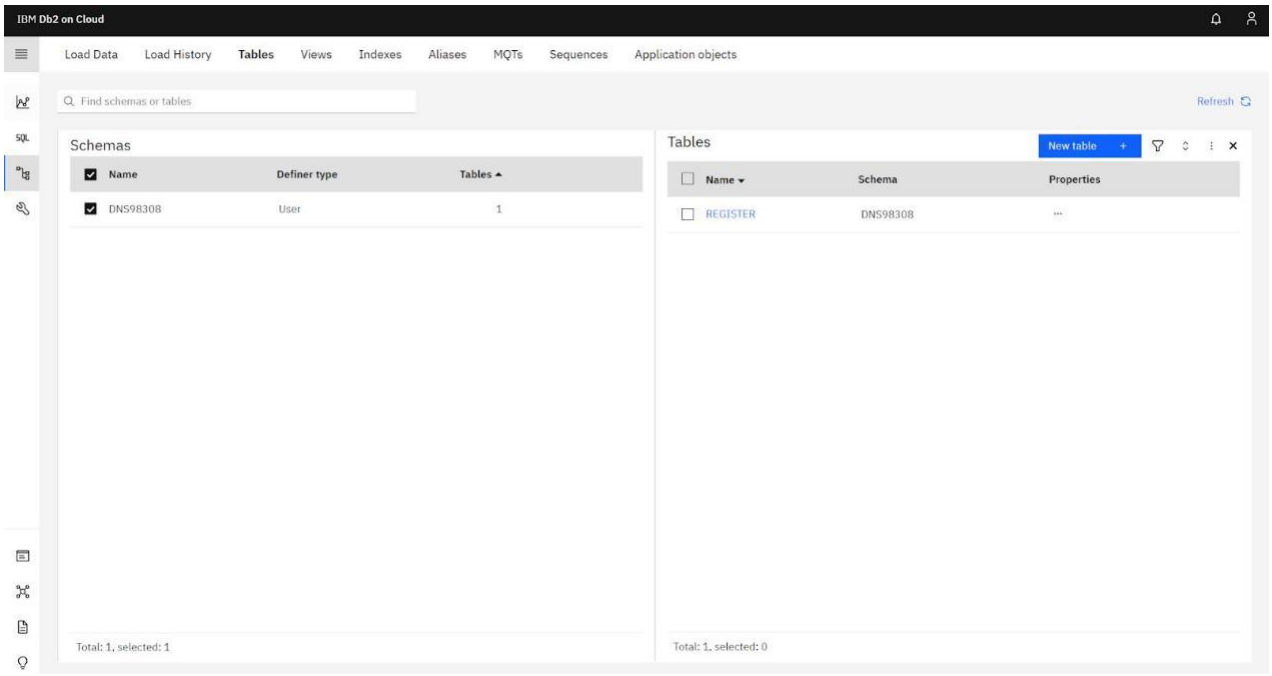
Now add all 2nd and all columns as the same as above.



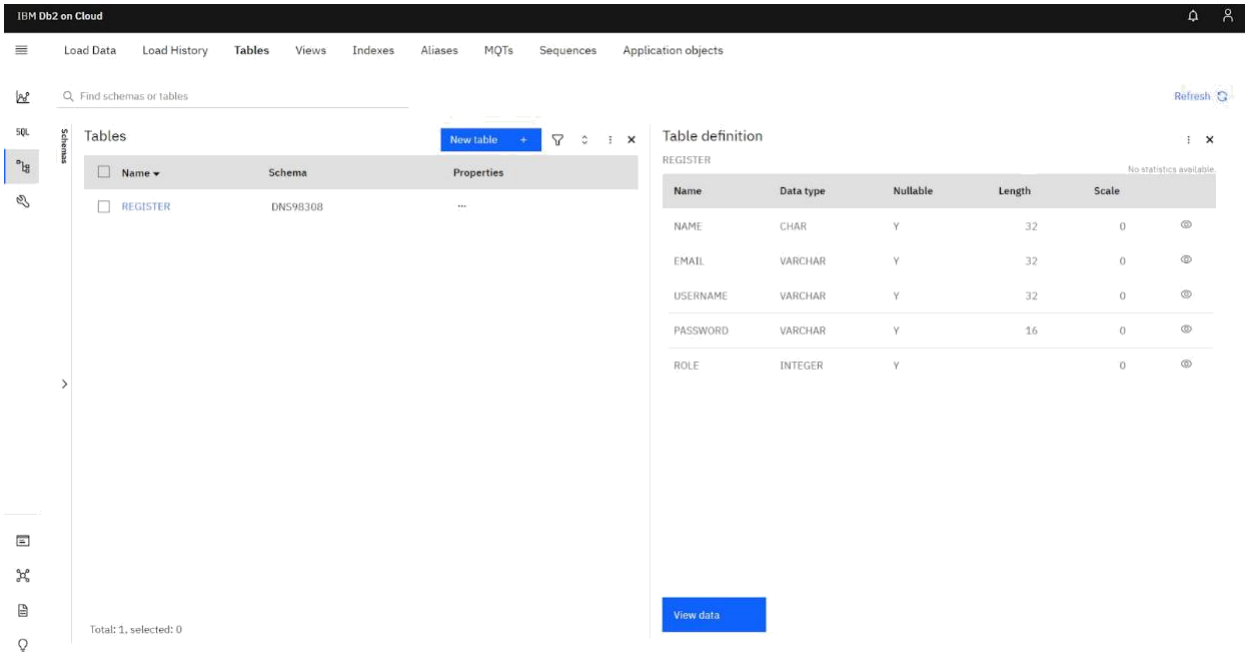
Once done adding all the columns along with the other variables, Now click on the create button on the right side bottom.



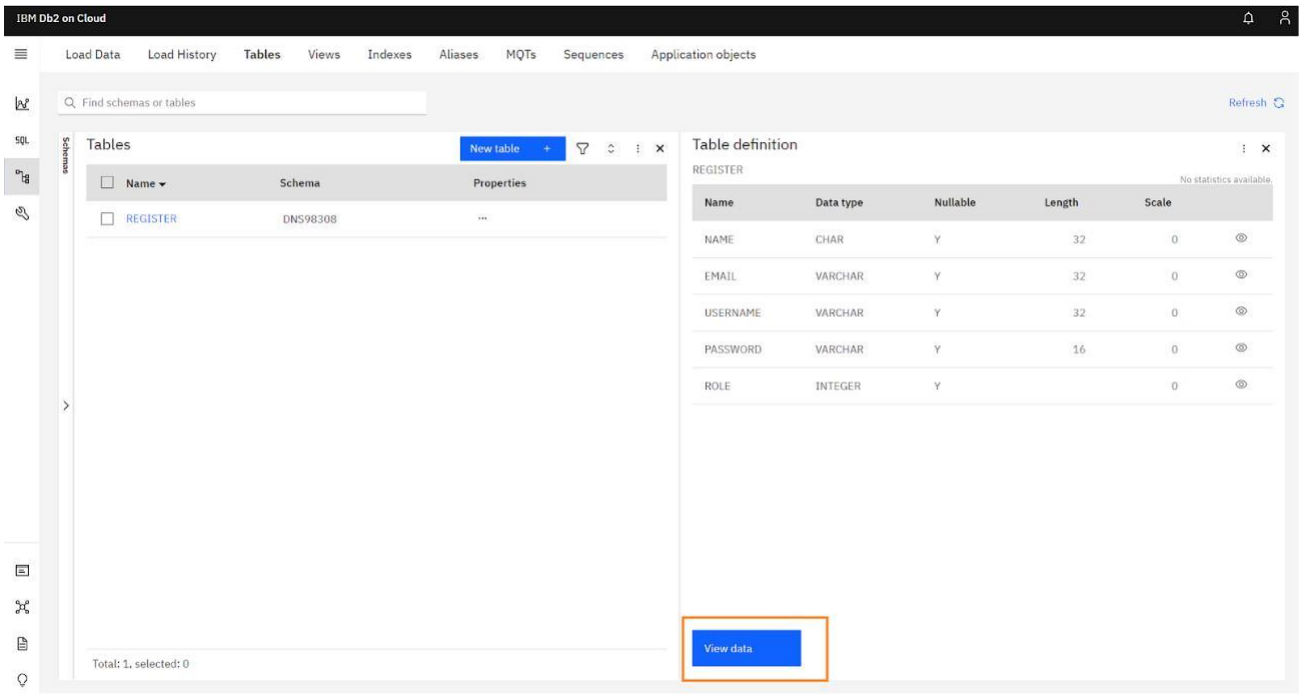
It will create a table.

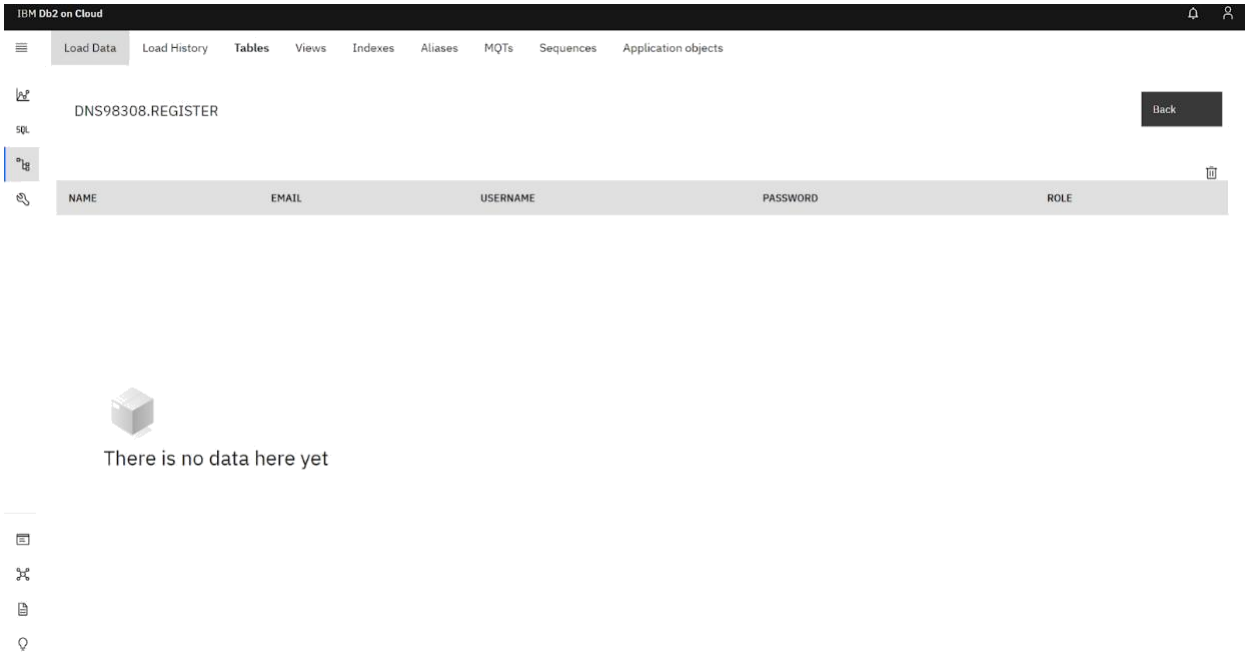


Click on the name register to see the table that we have created just now.



One Right side we can see all the columns and its properties that we have added just now. If you click on the view data button at the bottom you will see that data in the table.



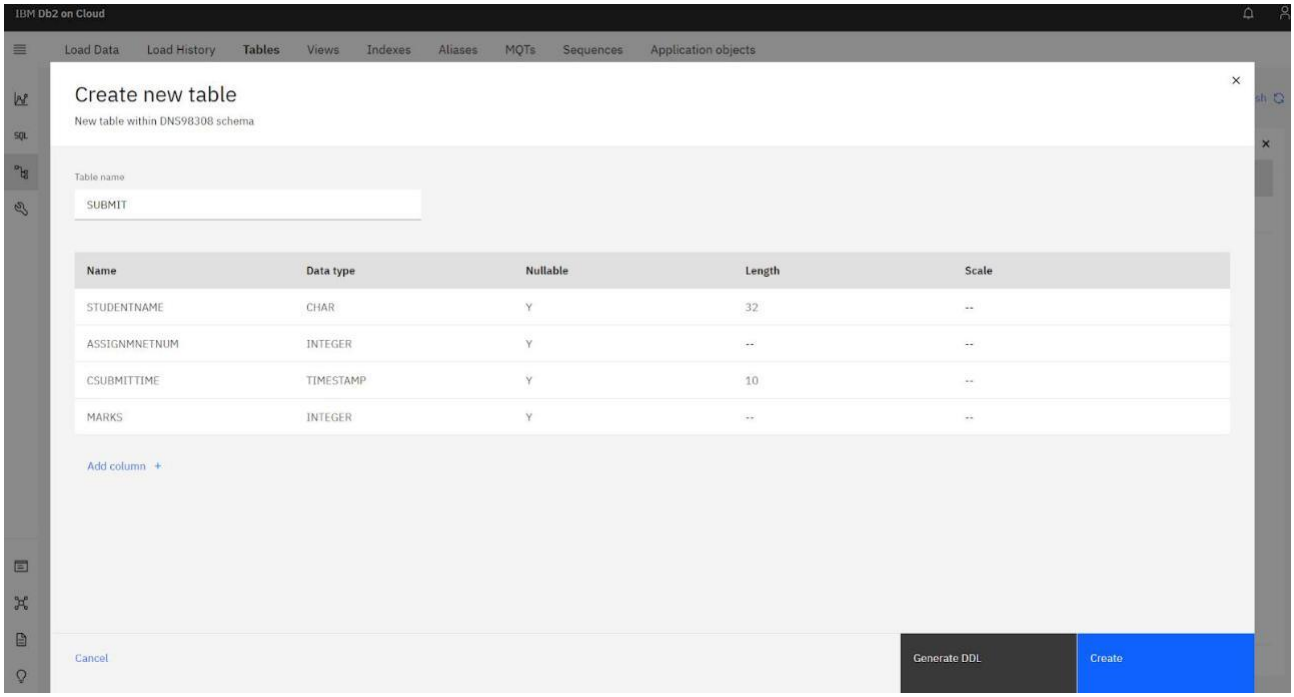


As of now we have not added any data to the table so it is empty.

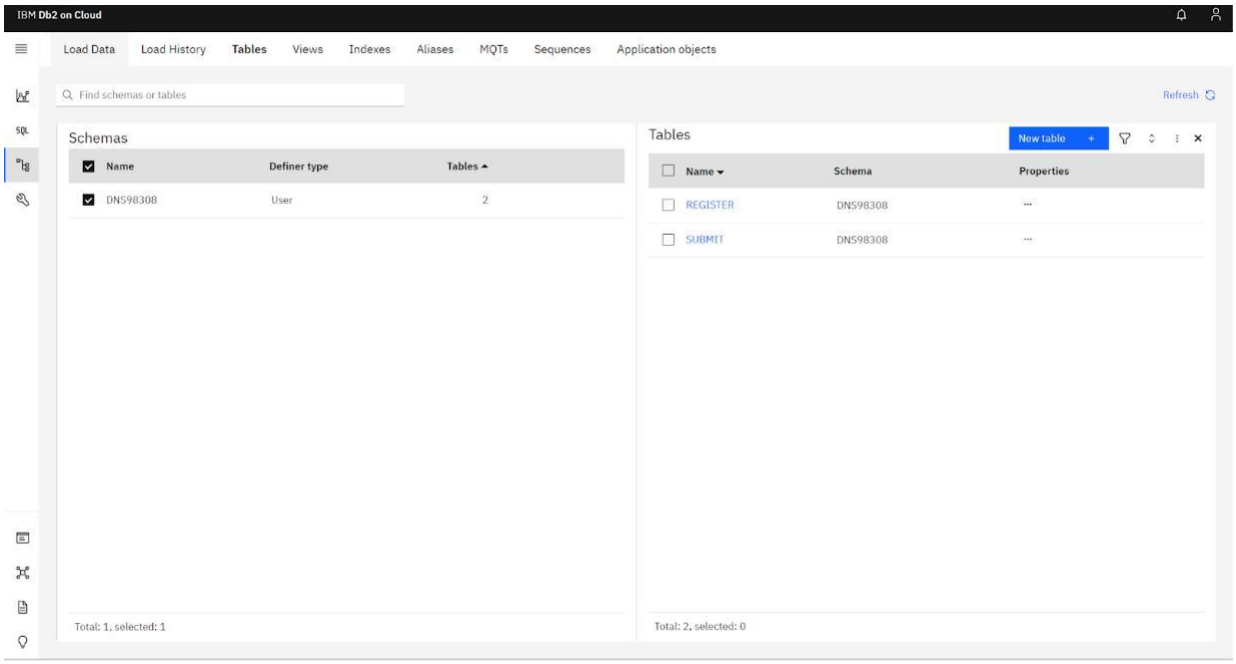
Activity 1.2: Creating Submit Table

In the same process as how we create a Register table , we will also create a submit table.

In the submit table we will have columns like studentname, assignmentnum, submittime, marks.



Now click on create in the bottom right corner it, will create a new table with these columns.

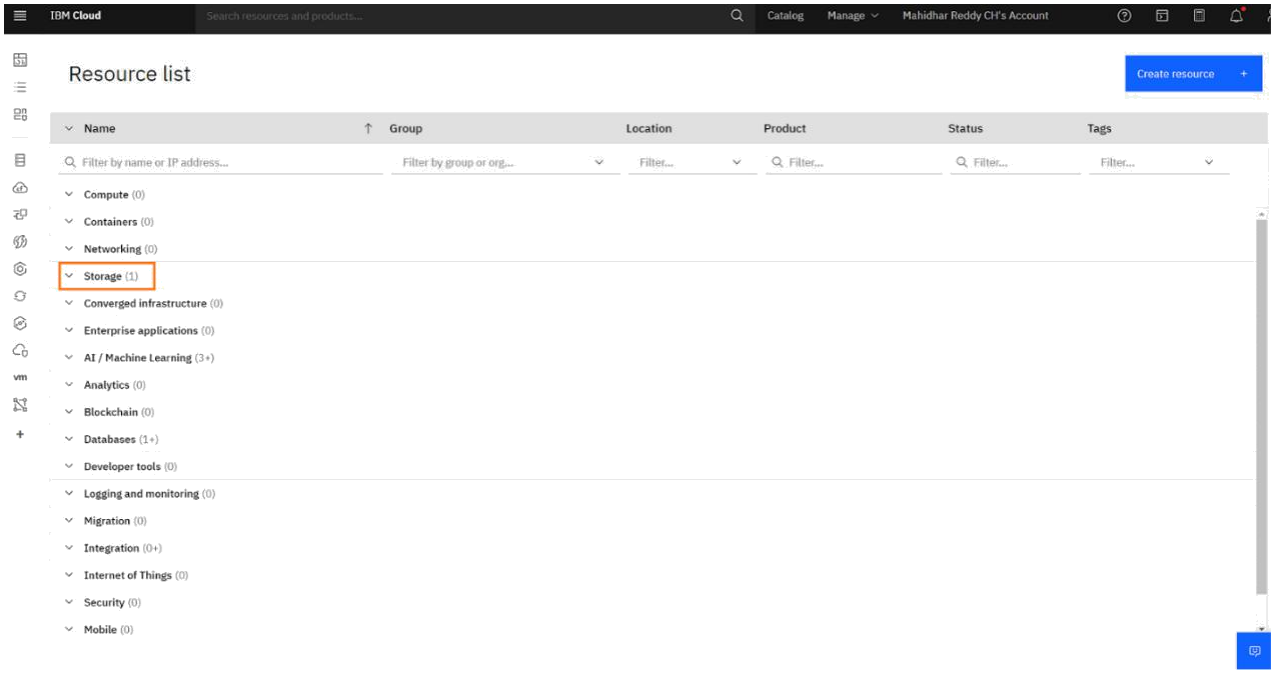


As you can see as of now we have created two tables. If we need more, we will create them in the next process.

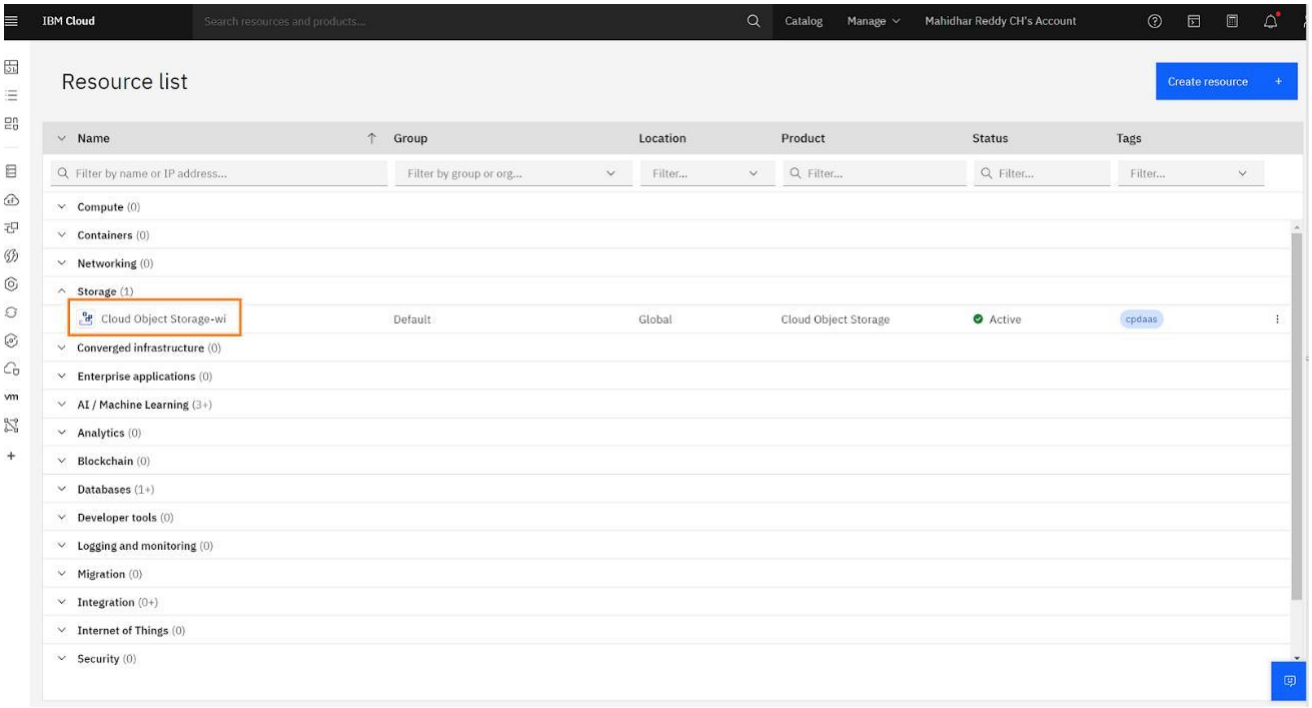
3.1

Creating A Object Storage

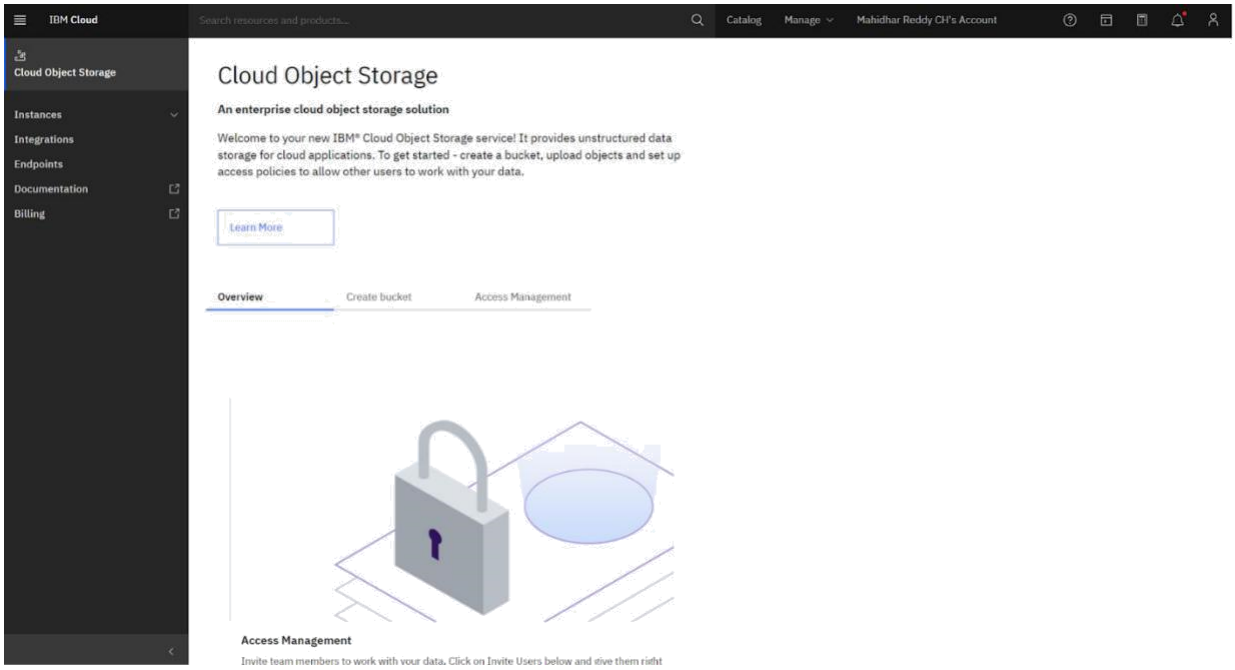
When you create an Object storage service, you will be redirected to the resources list. In that click on storage.



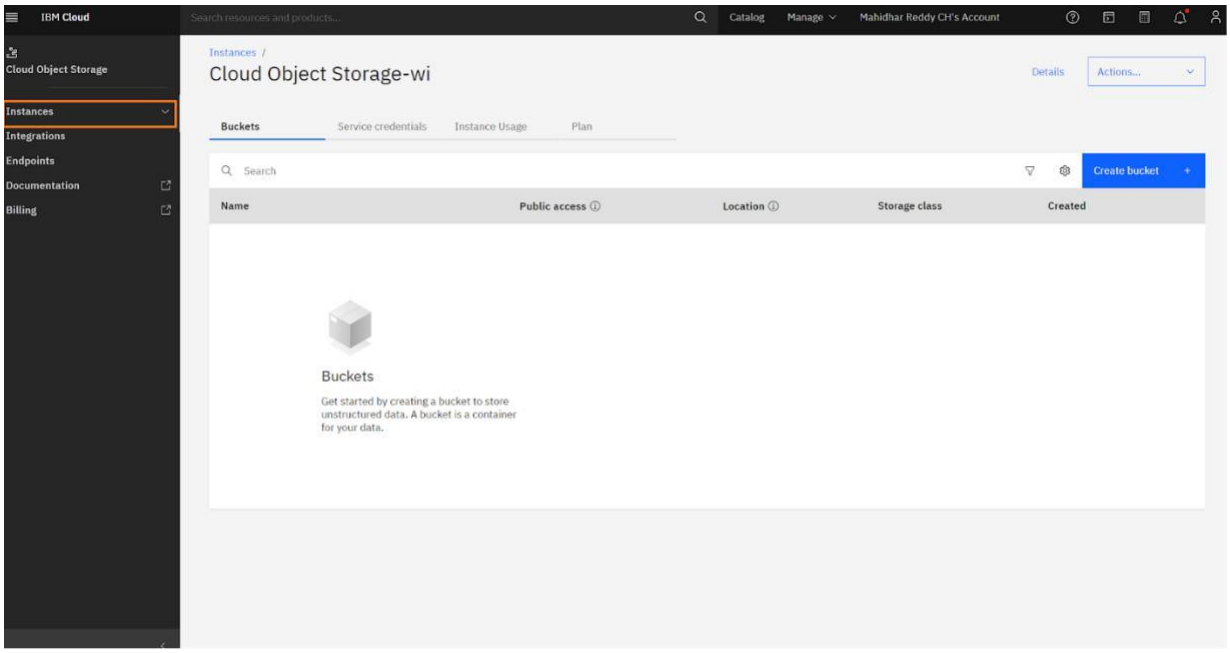
Now it will expand and show you the created object storage.



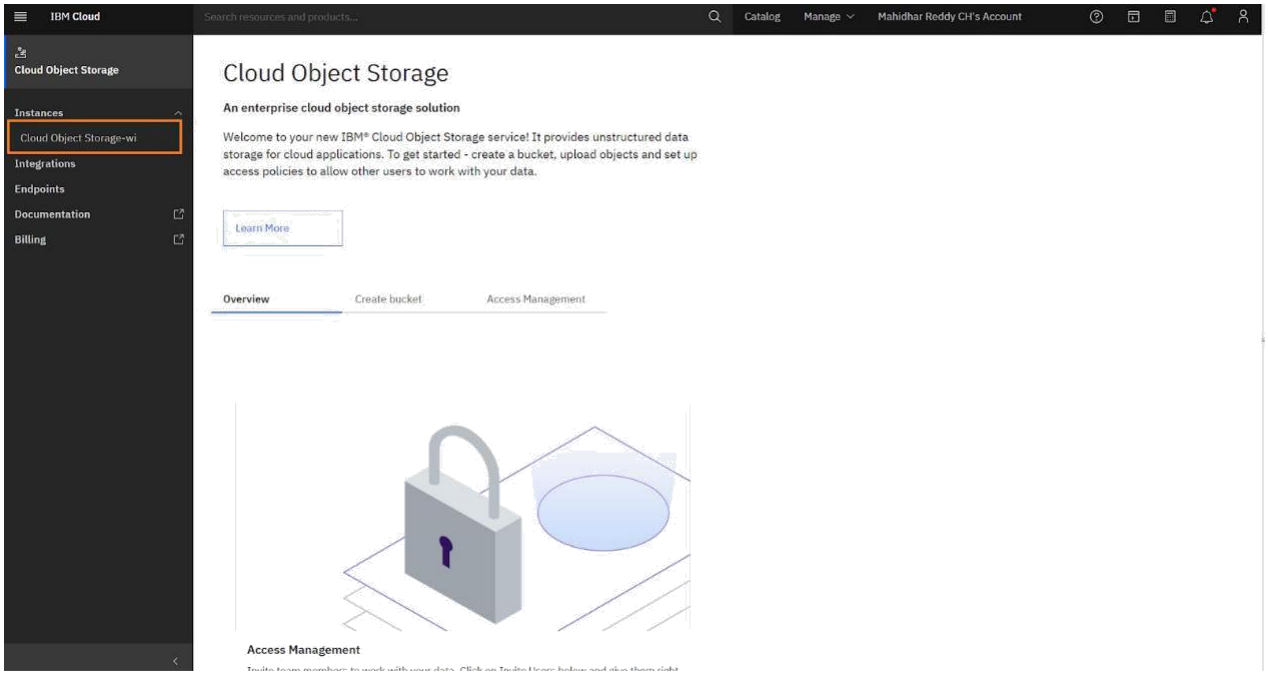
Now click on the name cloud object storage it will open the object storage ui in the new tab.



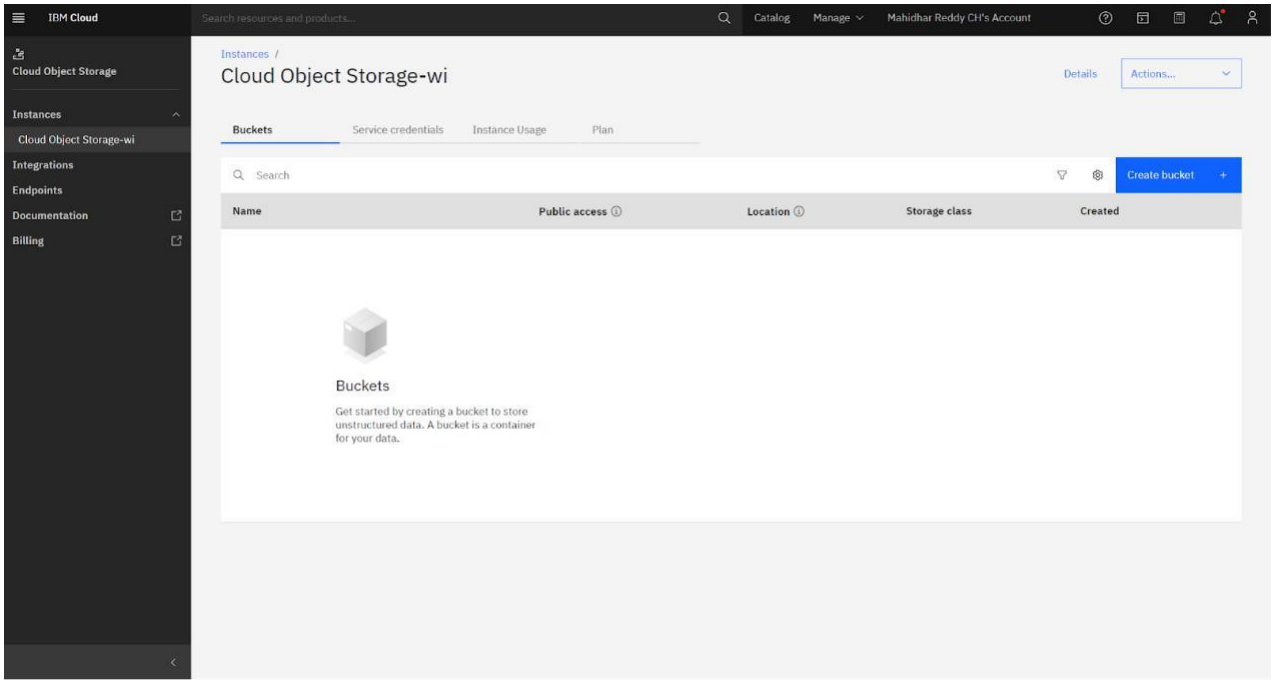
As just now we have created this, Nothing much is there in here. Now click on that Instances dropdown on the left side.



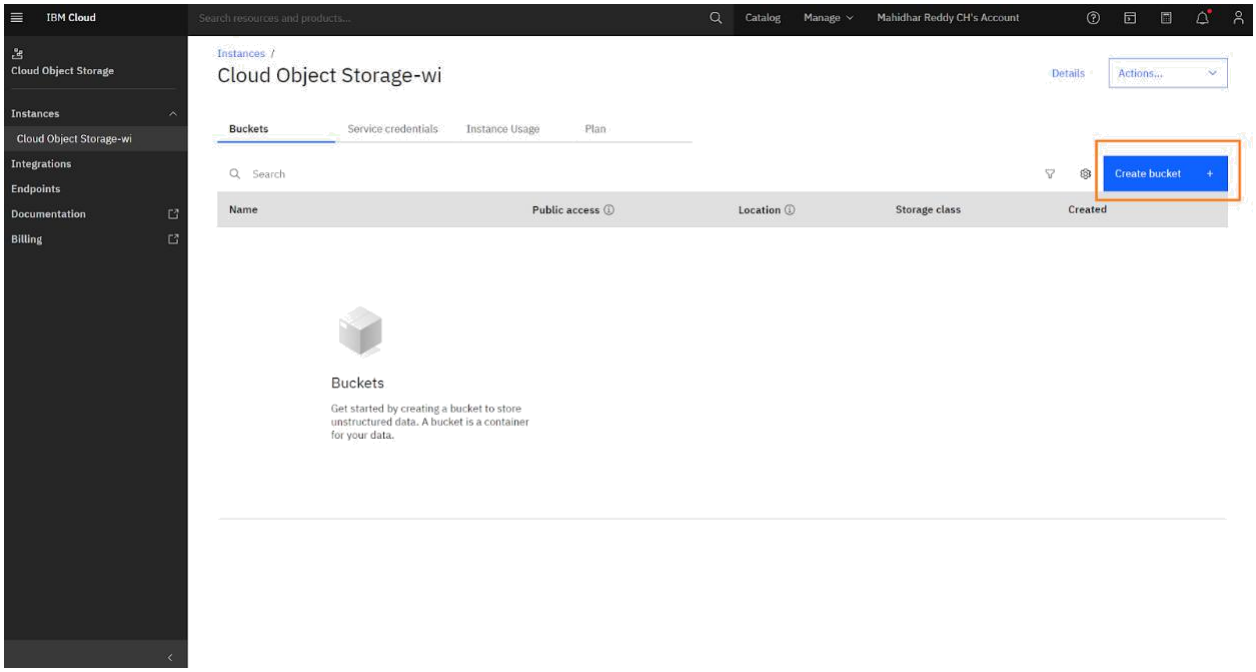
You can see Cloud object storage name, click on it.



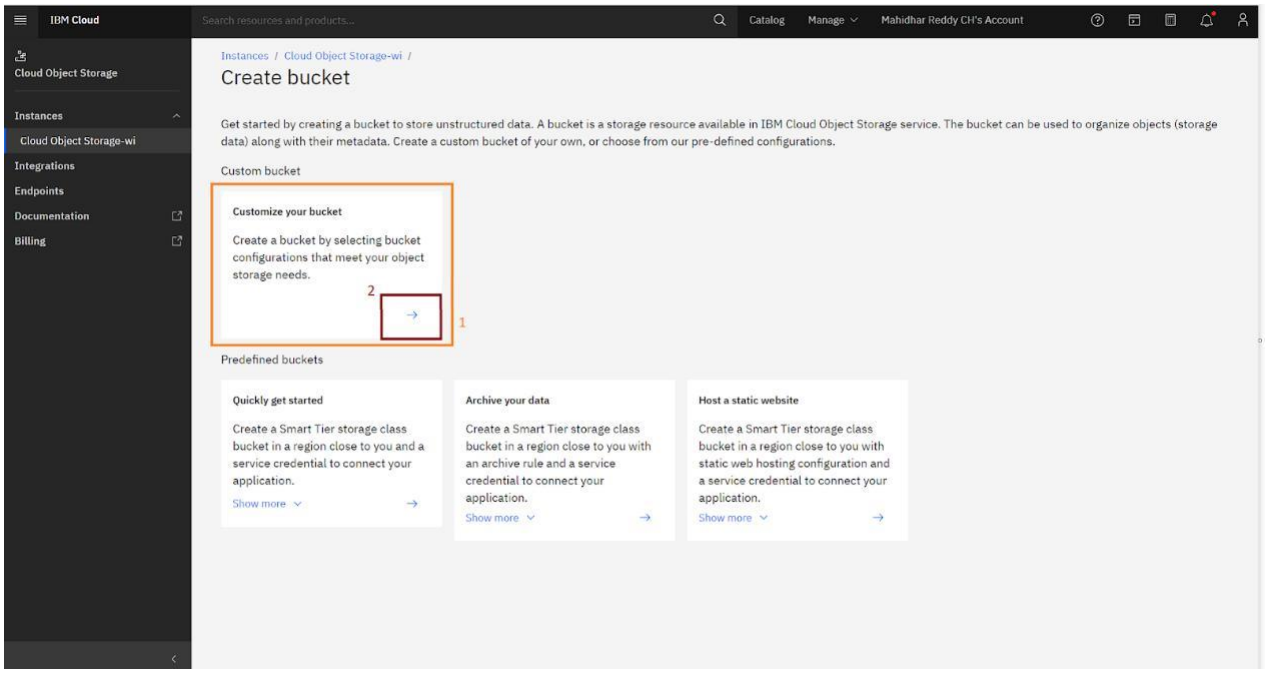
When you click on it, you can see a new tab opens on the right.



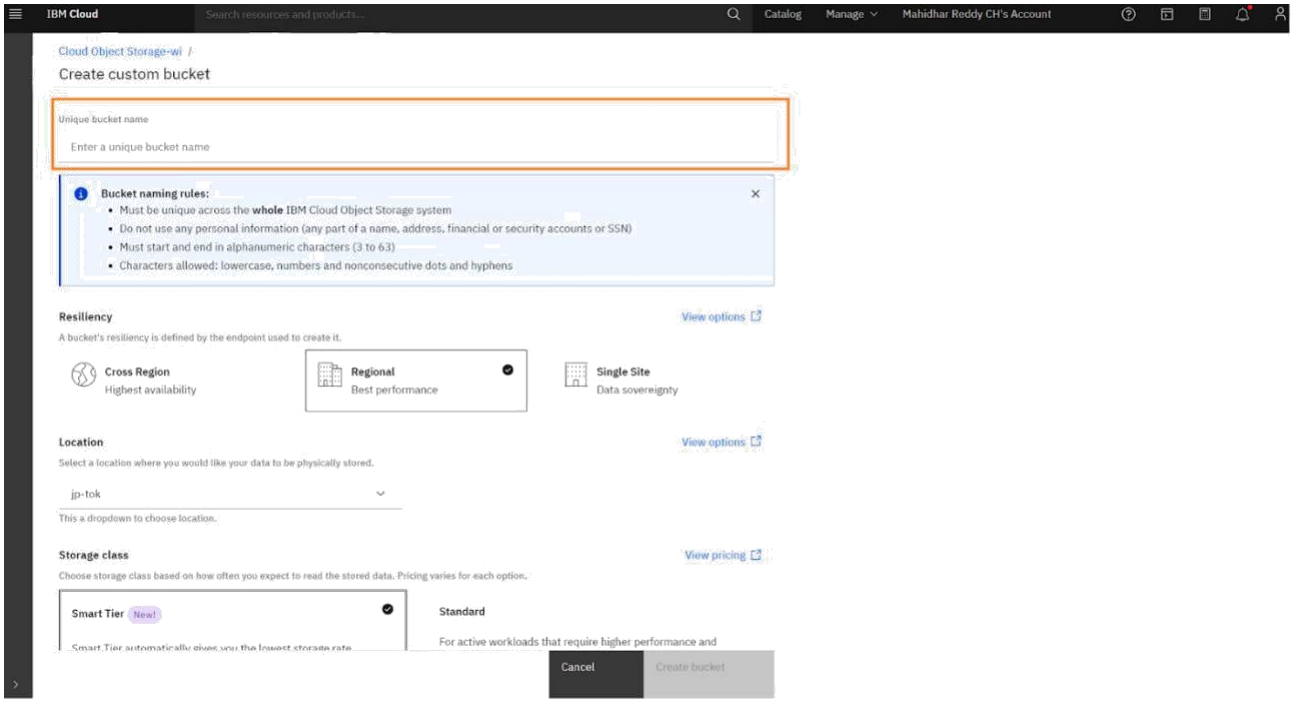
As of now we don't have anything, now click on that create bucket button on the right side top.



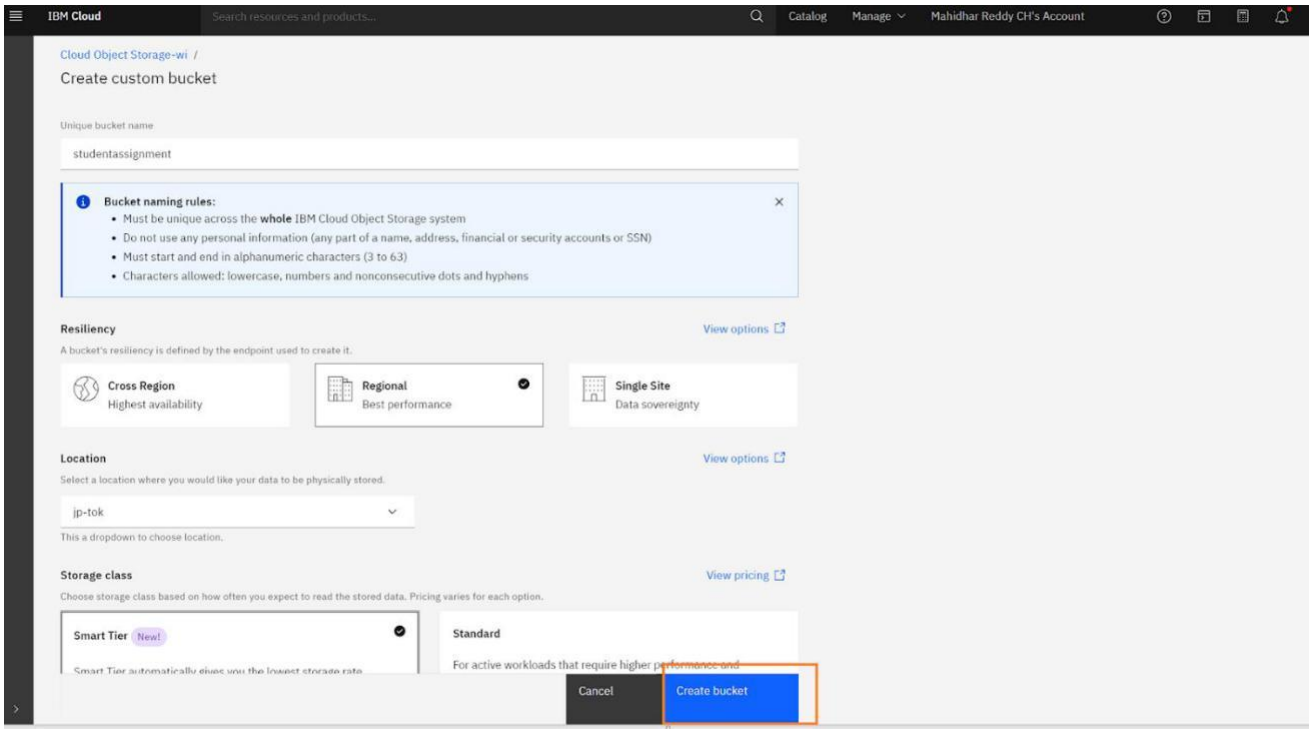
Now select the Customise your bucket arrow mark (2) to create a new bucket that can store the objects.



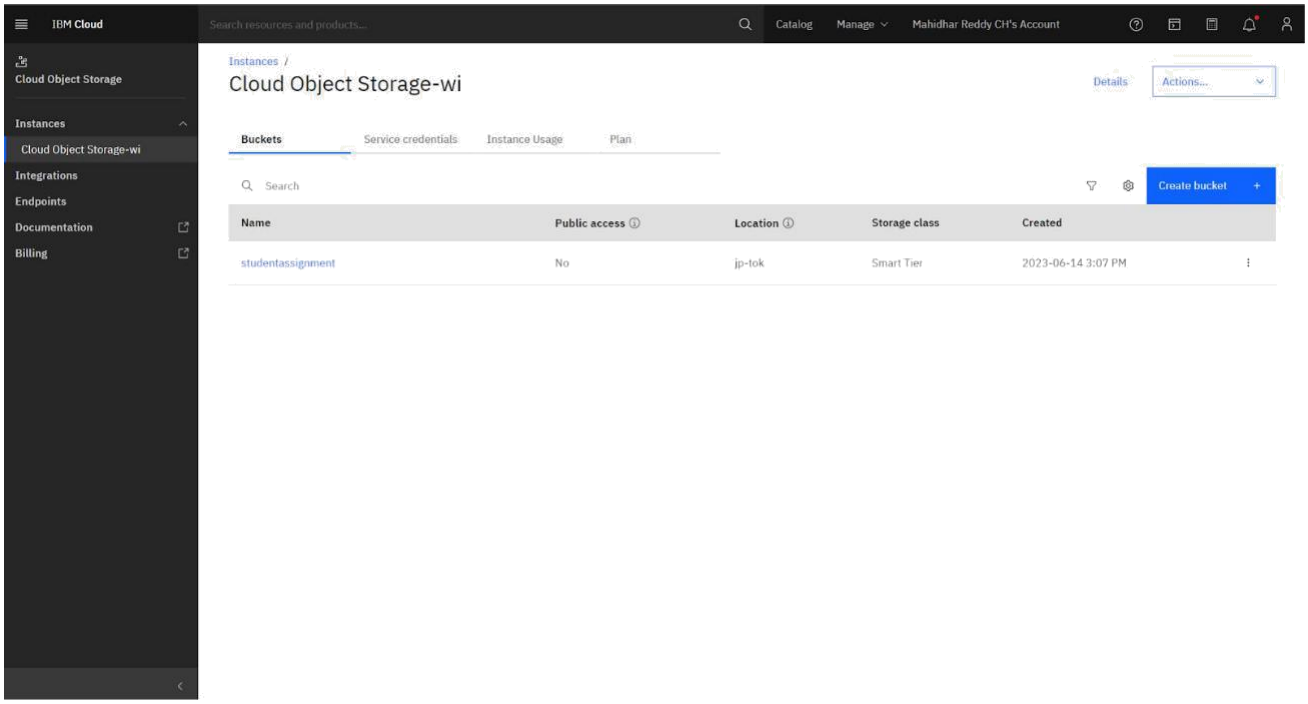
Now select on the text box, to name your bucket .



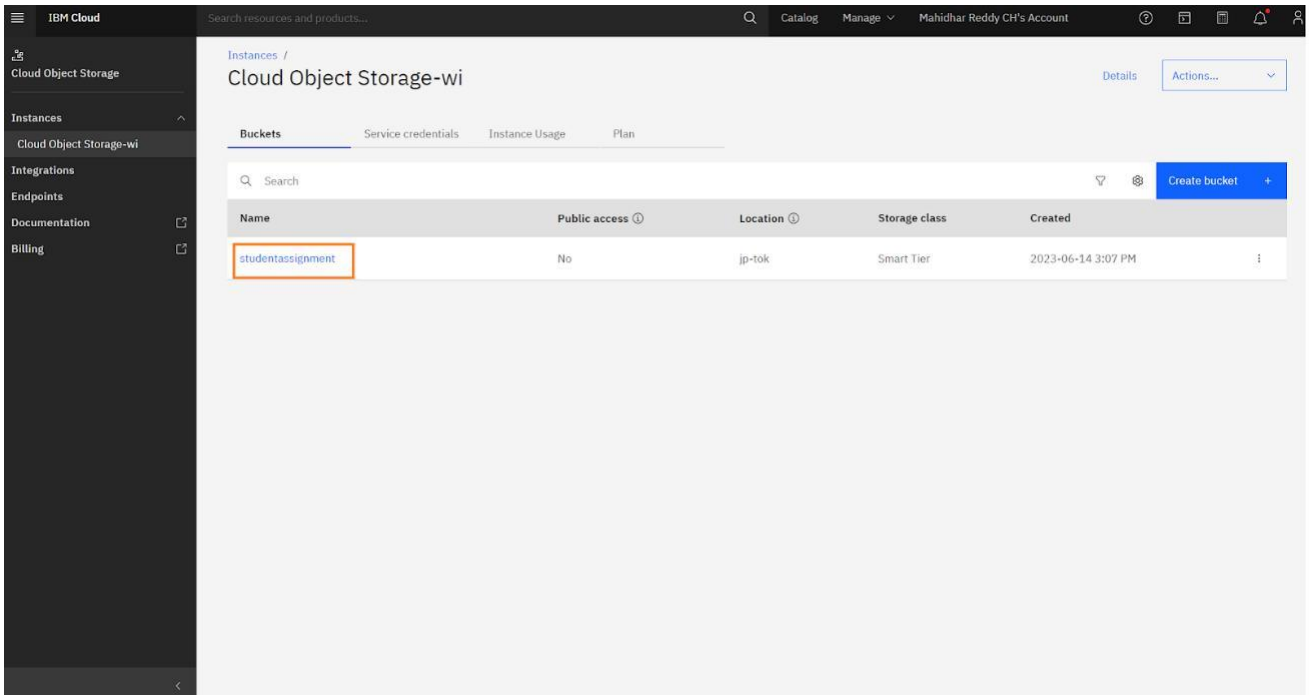
Give a unique name to the bucket (preferable to give the project or task files you are saving)



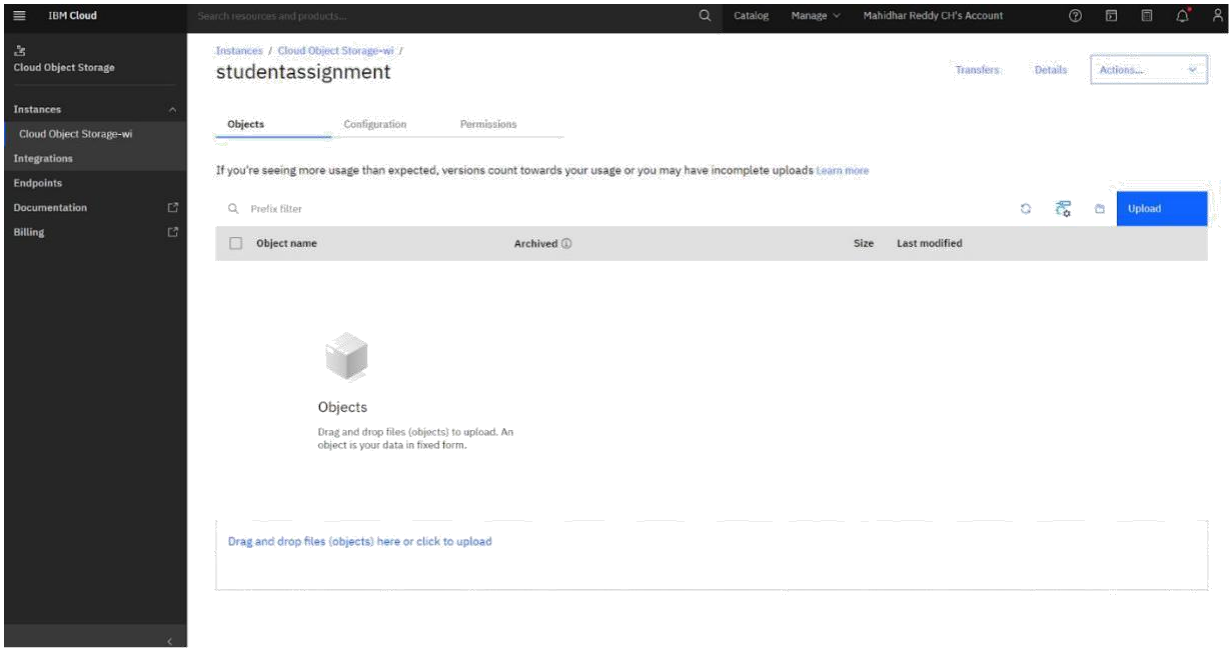
Now click on the the create bucket button in the right bottom corner to create a bucket



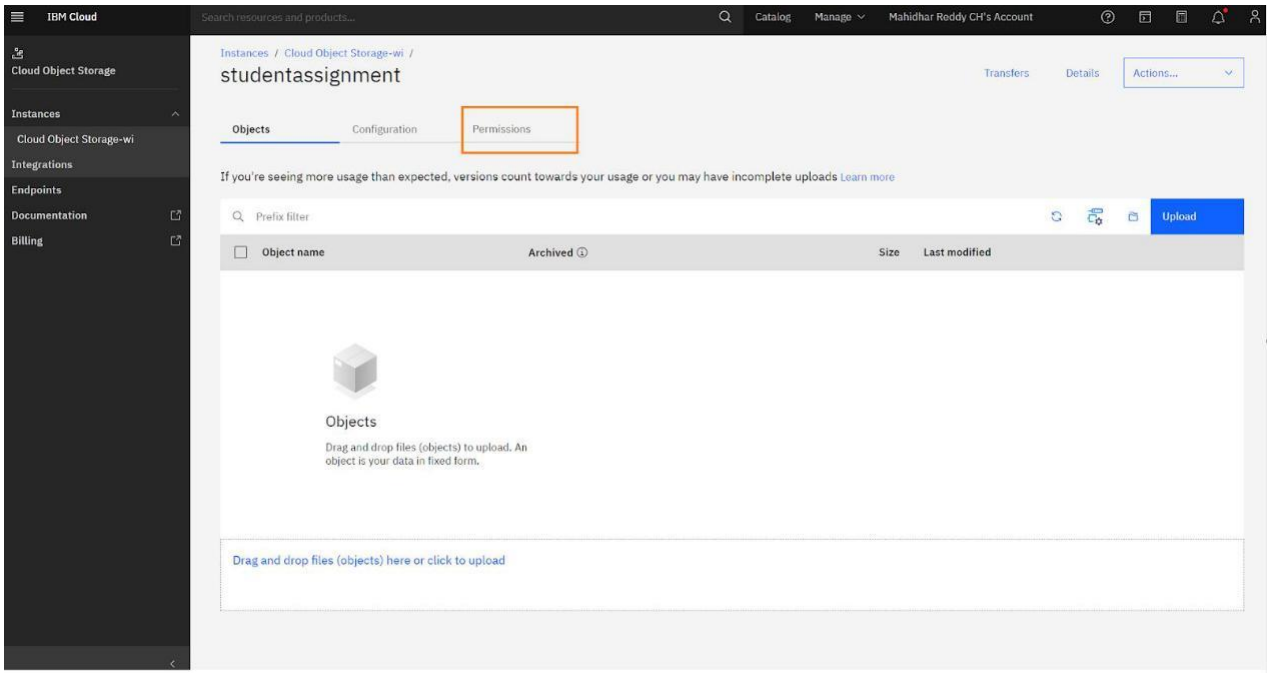
Now click on the bucket name which we have given just now.



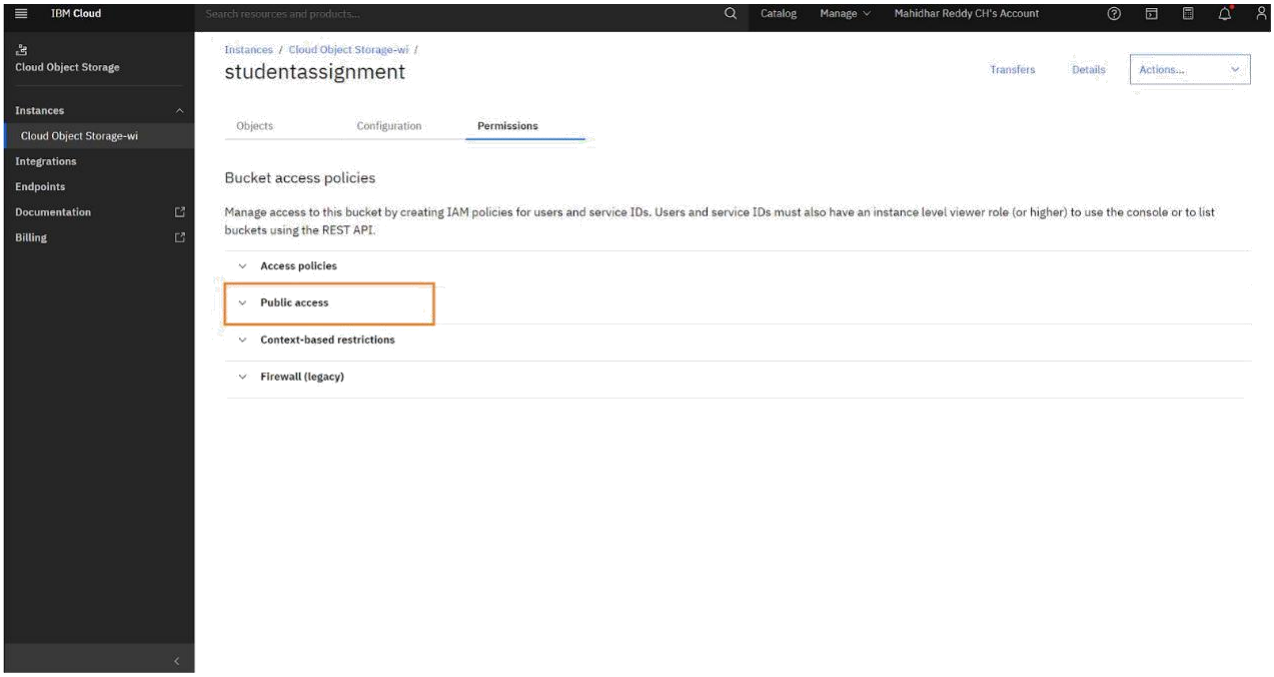
It will open a space where we can upload the files or where we can see those files if it is there.



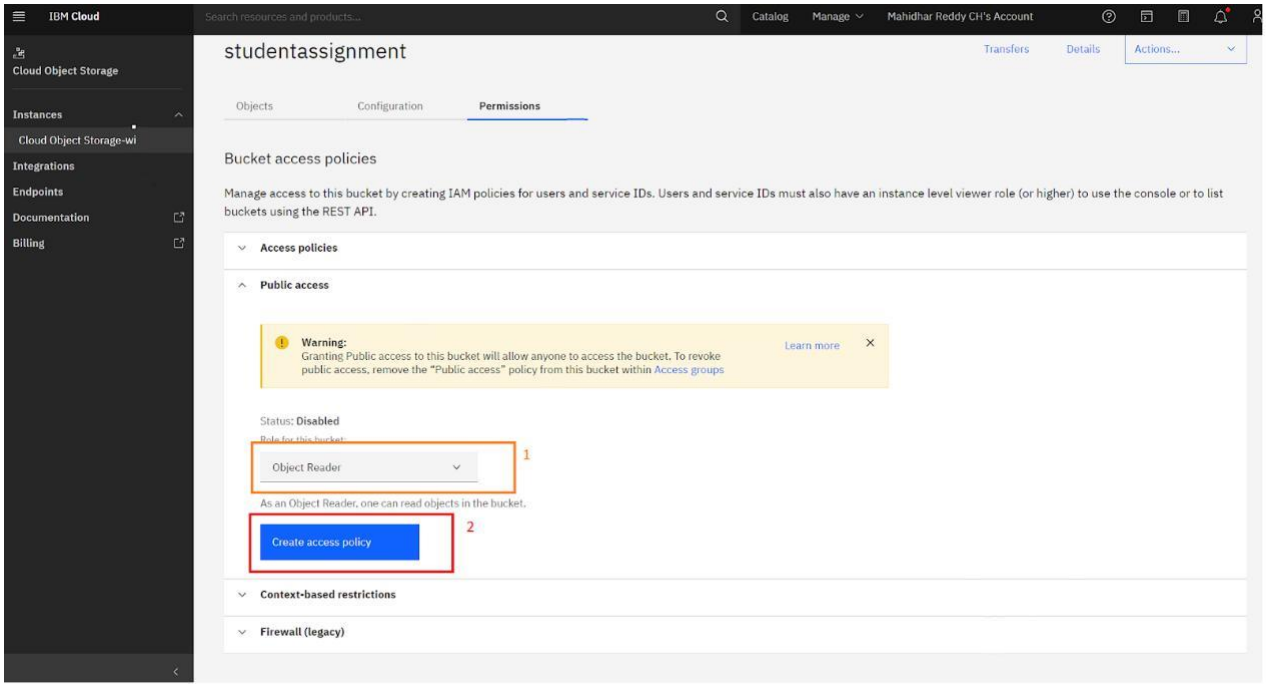
Now go to the permissions tab by clicking on it.



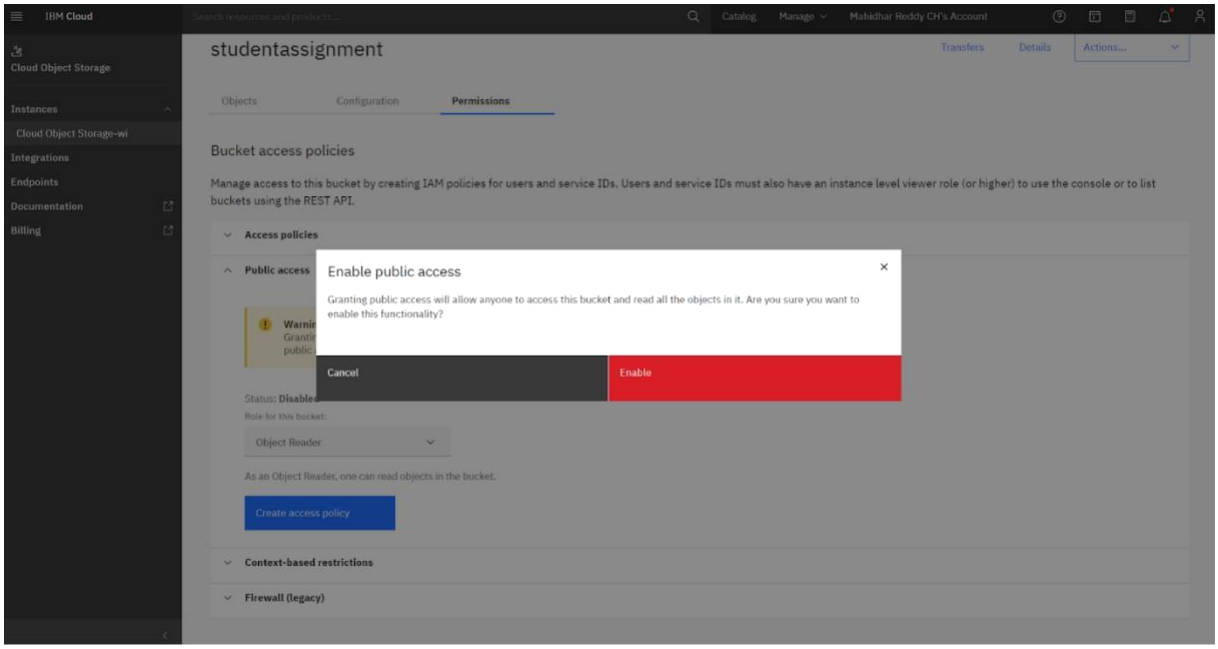
Now click on the public access drop down to change the permission settings.



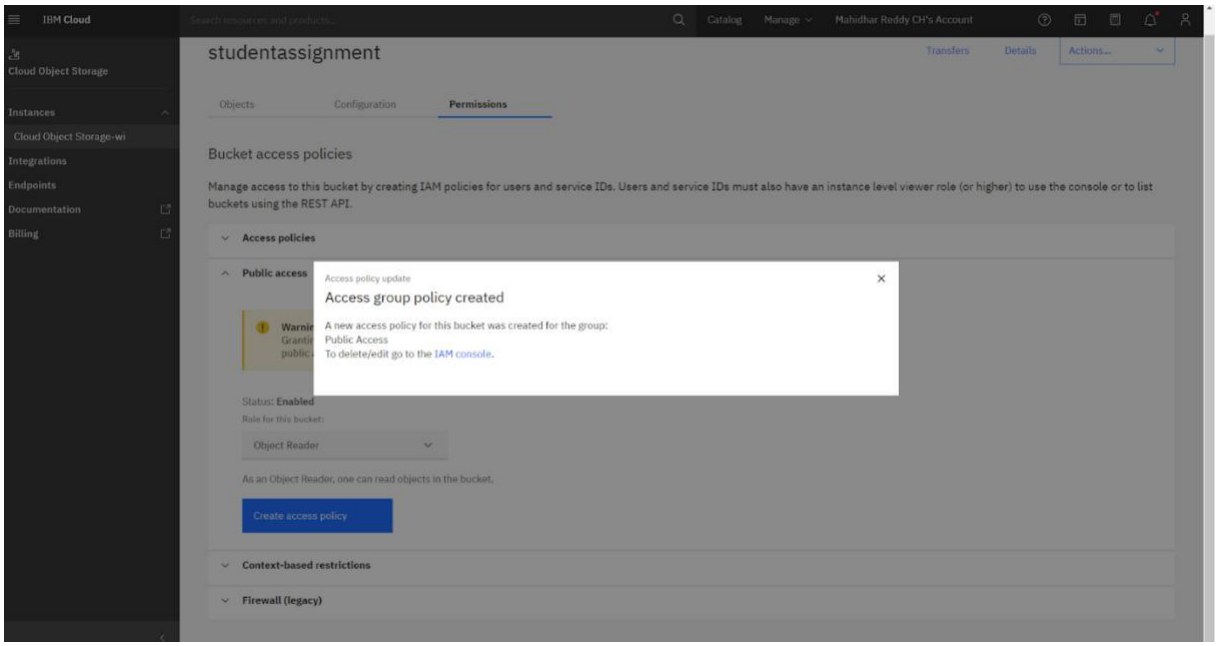
Now click on the Content Reader dropdown and select object reader (1). Now click on create access policy (2).



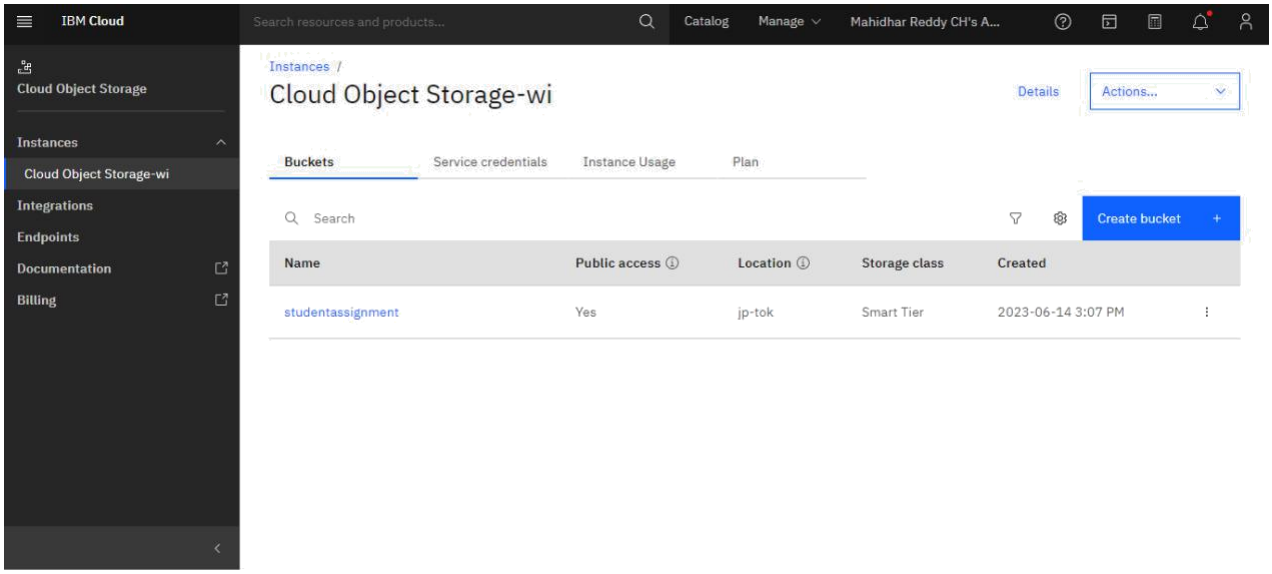
Now click on enable to change the permissions.



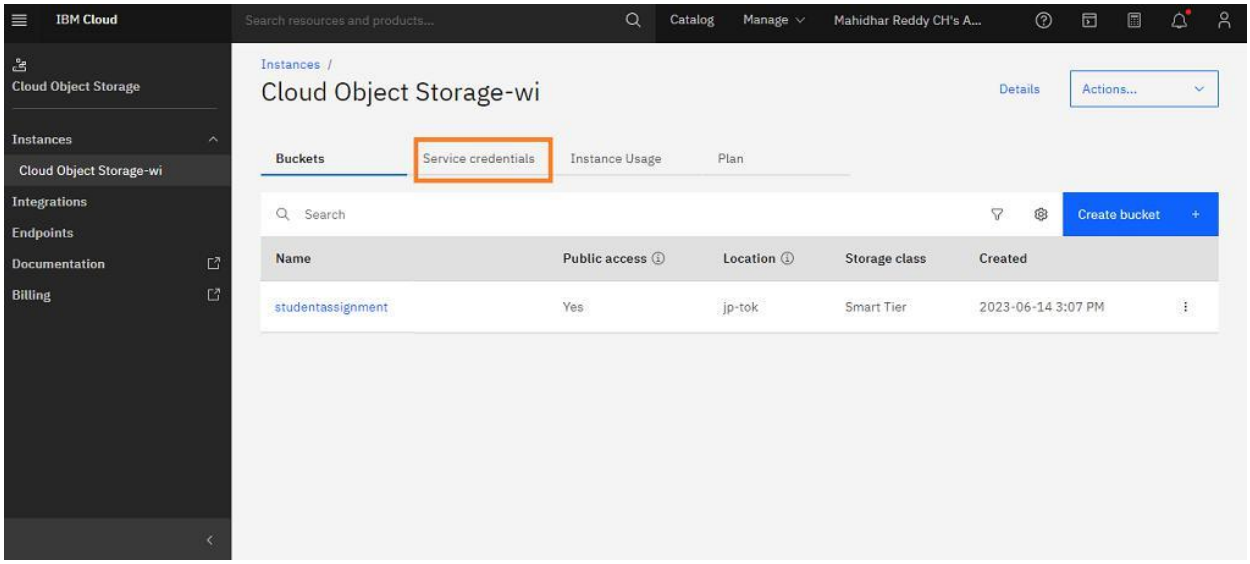
Once successful you will get a pop up. Click on cross mark(X).



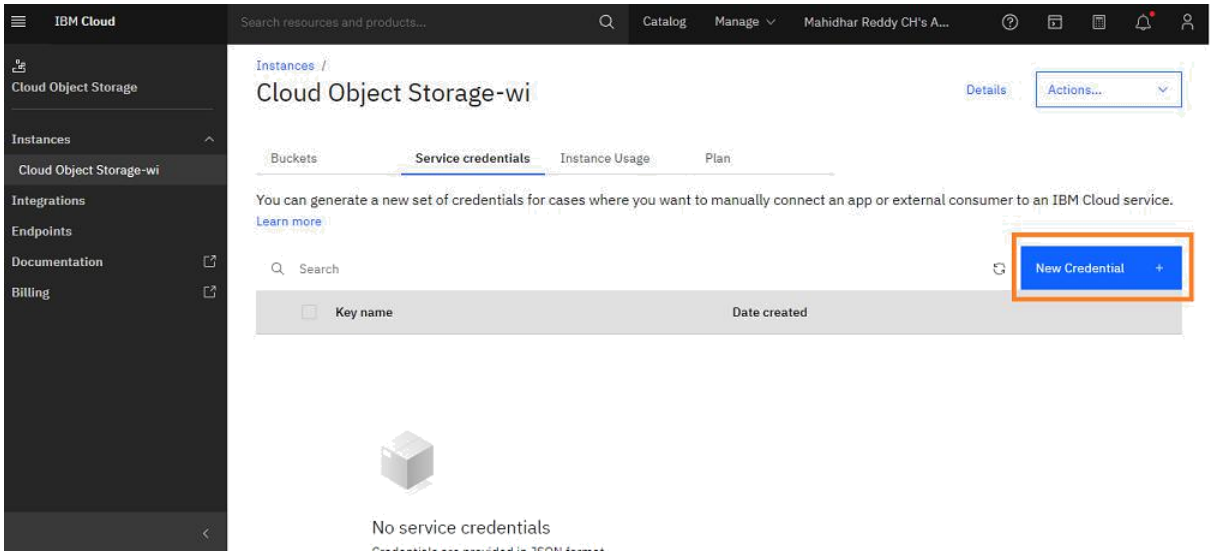
Now click on cloud object storage on the left side panel, in the dropdown of instances. So you will be redirected to the below page.



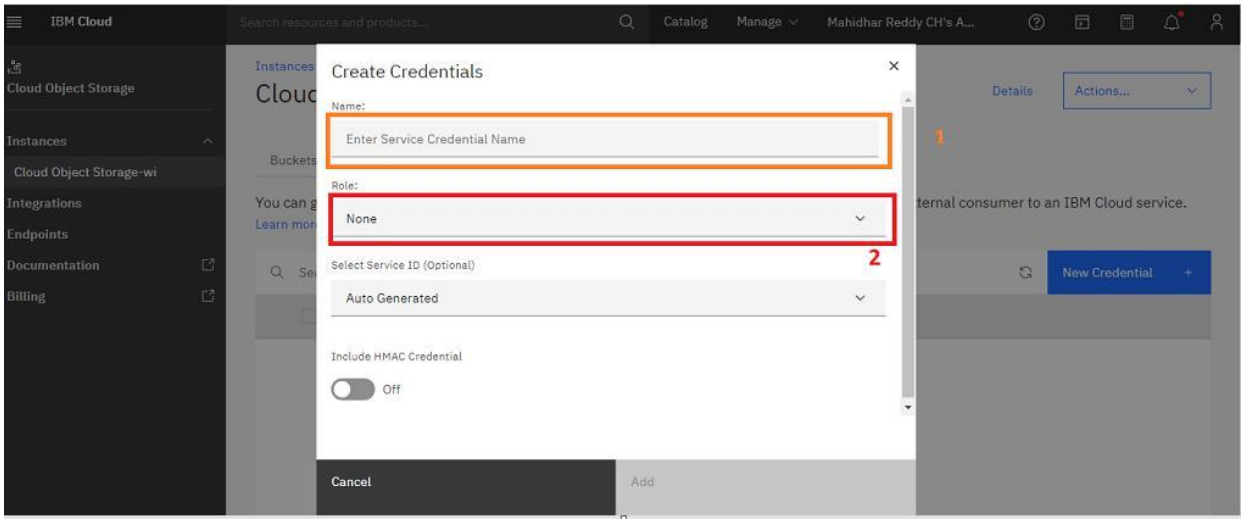
Now again click on the service credentials tab.



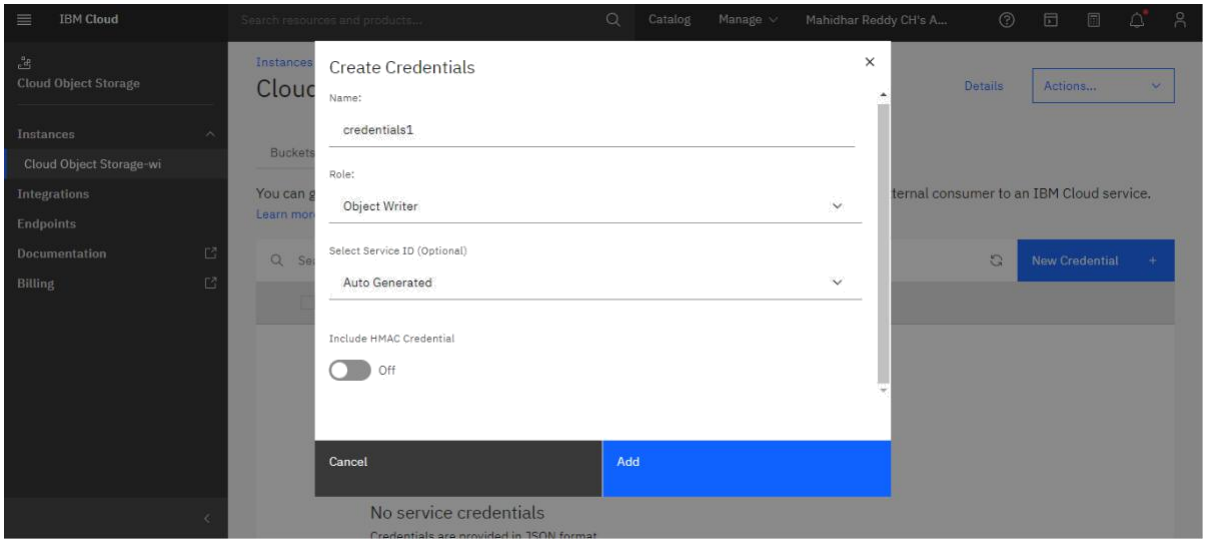
Now click on the New Credentials tab to create service credentials that are required to connect the object storage remotely.



Once you click on it, you will get a popup to add credentials .



In this, we will give the name of the credentials in the name (1) and we will select the role in the dropdown (2).

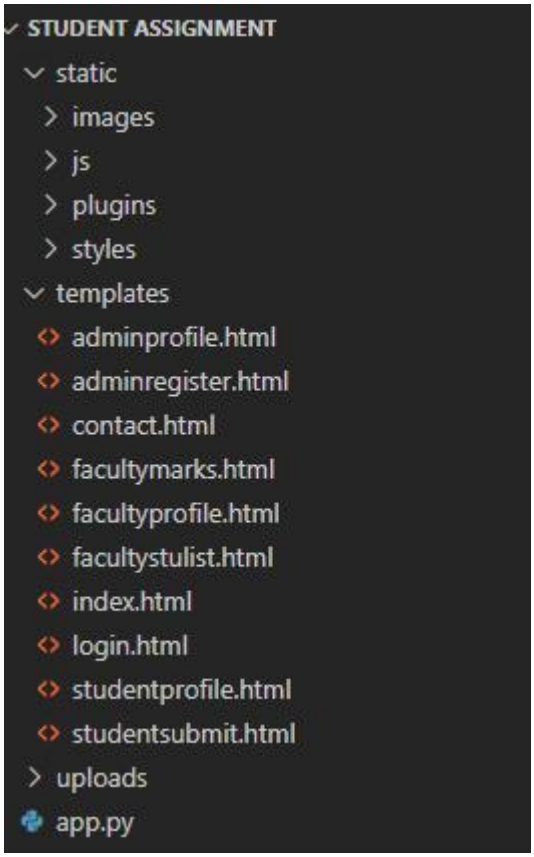


Give the name as credential 1 (we can give any name) and the role as Object Writer (Because we will read and write the objects both. Now click on Add in the bottom.

This is all about Object storage as of now. We will come back to this again in the next stages.

3.2 Creating Flask File

While we are doing flask, it would be good if we follow the folder structure that sort of flask defines.



There will be a main folder or project folder, here it is Student Assignment inside that we will have a few more folders like static, templates and uploads.

- static: Inside this folder, we will have all the static files like CSS, Javascript, and Images in their respective folders.
- templates: In this folder, we will have all the HTML templates/pages
- uploads: In this folder, we will store all the files uploaded by students to upload it to Object Storage.
- app.py: This is the flask file, in which we will have all the flask code.

3.3

Importing All Necessary Libraries, Modules And Initializing Flask Objects.

First, we will import all the necessary libraries and initialize the flask app object. That we will define how the application has to run on the server.

```python
from flask import Flask, render_template, request, session
import ibm_db
import ibm_boto3
from ibm_botocore.client import Config, ClientError
import os
import re
import random
import string
import datetime
import requests


app = Flask(__name__)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")
```

3.4

Rendering Index.Html Page

The route() decorator is to tell Flask what URL should trigger our function. To render a template you can use the render_template() method. All you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments.

```python
@app.route("/")
def index():
    return render_template("index.html")
```

3.5

Rendering Contact.Html Page

```
@app.route("/contact")
def contact():
    return render_template("contact.html")
```

3.6

Rendering Login.Html Page

```
conn = ibm_db.connect("DATABASE=bludb;
                      HOSTNAME=< Enter Your Host Name> ; PORT=<Port Number>; UID=<Enter Your Username>;
                      PASSWORD=<Enter Your Password>; SECURITY=SSL;
                      SSLServerCertificate = DigiCertGlobalRootCA.crt", "", "")
```

```python
@app.route("/login", methods=['POST','GET'])
def loginentered():
    global Userid
    global Username
    msg = ''
    if request.method == "POST":
        email = str(request.form['email'])
        print(email)
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND PASSWORD=?"  # from db2 sql table
        stmt = ibm_db.prepare(conn, sql)
        # this username & password is should be same as db-2 details & order also
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            Userid = account['EMAIL']
            session['email'] = account['EMAIL']
            Username = account['USERNAME']
            Name = account['NAME']
            msg = "logged in successfully !"
            sql = "SELECT ROLE FROM register where email = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, email)
            ibm_db.execute(stmt)
            r = ibm_db.fetch_assoc(stmt)
            print(r)
            if r['ROLE'] == 1:
                print("STUDENT")
                return render_template("studentprofile.html", msg=msg, user=email, name = Name, role= "STUDENT", username=Username, email = email)
            elif r['ROLE'] == 2:
                print("FACULTY")
                return render_template("facultyprofile.html", msg=msg, user=email, name = Name, role= "FACULTY", username=Username, email = email)
            else:
                return render_template('adminprofile.html', msg=msg, user=email, name = Name, role= "ADMIN", username=Username, email = email)
        else:
            msg = "Incorrect Email/password"

        return render_template("login.html", msg=msg)
    else:
        return render_template("login.html")
```

In this snippet of code, as we can see we will render the login.html template (in the last line of code). When the user enters their details, those details are collected by using the form. That form is action to "/login". The same route we use here. We will get the details and

based on those details we will validate the login. If the details are correct. According to the role, it will redirect them to their respective pages.

The login email and username of the user we will track by using the session object of flask to track user activities.
Note: The Picture with line 14, 15, 16 and 17 should be in a single line.

3.7

Rendering Adminprofile.Html

```
31  @app.route("/adminprofile")
32  def aprofile():
33      return render_template("adminprofile.html")
```

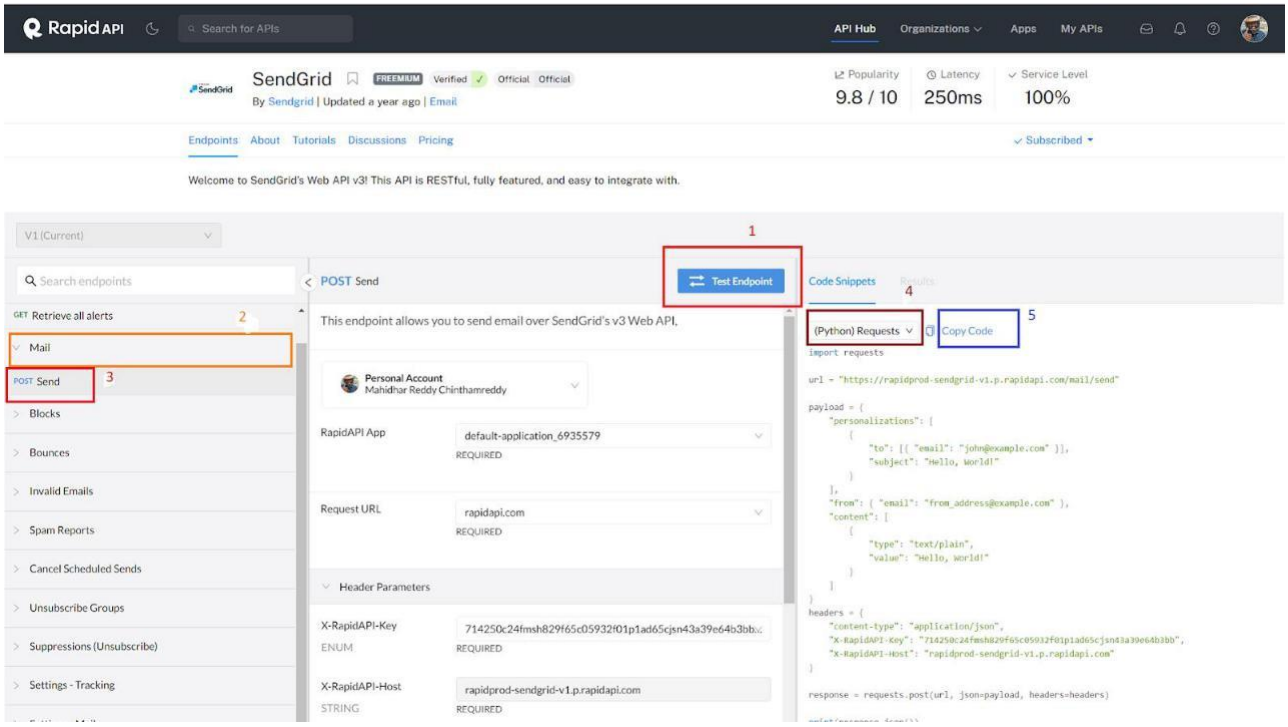
3.8

Rendering The Adminregister.Html

```python
@app.route("/register", methods=['POST', 'GET'])
def signup():
    msg = ''
    if request.method == 'POST':
        name = request.form["sname"]
        email = request.form["semail"]
        username = request.form["susername"]
        role =int(request.form['role'])
        password = ''.join(random.choice(string.ascii_letters) for i in range(0,8))
        link = 'https://university.ac.in/portal'
        print(password)
        sql = "SELECT* FROM register WHERE email= ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = "Already Registered"
            return render_template('adminregister.html', error=True, msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address!"
        else:
            insert_sql = "INSERT INTO register VALUES (?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            # this username & password is should be same as db-2 details & order also
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 5, role)
```

```
            ibm_db.bind_param(prep_stmt, 4, password)
            ibm_db.execute(prep_stmt)
            payload = {
                "personalizations": [
                    {
                        "to": [{"email": email}],
                        "subject": "Student Account Details"
                    }
                ],
                "from": {"email": "<Enter Your Sender Email Id"},
                "content": [
                    {
                        "type": "text/plain",
                        "value": """Dear {} ,  \n Welcome to University,
                        Here there the details to Login Into your student portal link : {} \n
                        YOUR Username : {} \n  PASSWORD : {}  \n Thank you \n Sincerely\n
                        Office of  Admissions\n Mahidhar Institute of Technology \n
                        E-Mail: admission@university.ac.in ; Website: www.university.ac.in"""
                        .format( name,link, username, password)
                    }
                ]
            }
            headers = {
                "content-type": "application/json",
                "X-RapidAPI-Key": "<Enter your Rapid api - api key>",
                "X-RapidAPI-Host": "rapidprod-sendgrid-v1.p.rapidapi.com"
            }
            response = requests.request("POST", url, json=payload, headers=headers)
            print(response.text)
            msg = "Registration Successful"
    return render_template('adminregister.html', msg=msg)
```

SendGrid Email service - Rapid API link: https://rapidapi.com/sendgrid/api/sendgrid



- If you login by creating the account in Rapid api, It will be like subscribing to test. On the next page you will be pricing details, click on the subscribe button under $0 (zero dollars).
- Now come back to the endpoints section click on the Mail dropdown to see the list of mail sending services they offer.
- Currently they are offering only one i.e. send. Click on it.
- Now by default the dropdown will be for (Node.js)Axios. Click on it and from the dropdown select the python. In that select requests

Now copy the code and paste it in the flask code. (which you see in the above screenshots).
Note: You need to customise the content section in the copy code after pasting, according to how you want your mail to be.

3.9

Rendering Studentprofile.Html

```
27  @app.route("/studentprofile")
28  def sprofile():
29      return render_template("studentprofile.html")
```

3.10

Rendering The Studentsubmit.Html

```python
@app.route("/studentsubmit", methods=['POST','GET'])
def sassignment():
    u = Username.strip()
    subtime = []
    ma = []
    sql = "SELECT SUBMITTIME, MARKS from SUBMIT WHERE STUDENTNAME = ? "
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, u)
    ibm_db.execute(stmt)
    st = ibm_db.fetch_tuple(stmt)
    while st !=False:
        subtime.append(st[0])
        ma.append(st[1])
        st = ibm_db.fetch_tuple(stmt)
    print(subtime)
    print(ma)
    if request.method=="POST":
       for x in range (1,5):
        x = str(x)
        y = str("file"+x)
        print(type(y))
        f=request.files[ y ]
        print(f)
        print(f.filename)



        if f.filename != '':

            basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
            #print("current path",basepath)
            filepath=os.path.join(basepath,'uploads',u+x+".pdf") #from anywhere in the system we can give image but we want that image l
            #print("upload folder is",filepath)
            f.save(filepath)
```

```python
            # connecting with cloud object storage

            COS_ENDPOINT = "<Enter your COS EndPoint>"
            COS_API_KEY_ID = "<Enter your COS API Key>"
            COS_INSTANCE_CRN = "<Enter your COS CRN Instance id>"
            cos = ibm_boto3.client("s3",ibm_api_key_id=COS_API_KEY_ID,ibm_service_instance_id=COS_INSTANCE_CRN,
                                   config=Config(signature_version="oauth"),endpoint_url=COS_ENDPOINT)
            cos.upload_file(Filename= filepath,Bucket='studentassignment',Key= u+x+".pdf")
            msg = "Uploding Successful"
            ts = datetime.datetime.now()
            t = ts.strftime("%Y-%m-%d %H:%M:%S")
            sql1 = "SELECT * FROM SUBMIT WHERE STUDENTNAME = ? AND ASSIGNMENTNUM = ?"
            stmt = ibm_db.prepare(conn, sql1)
            ibm_db.bind_param(stmt, 1, u)
            ibm_db.bind_param(stmt, 2, x)
            ibm_db.execute(stmt)
            acc = ibm_db.fetch_assoc(stmt)
            print(acc)
            if acc == False:
                sql = "INSERT into SUBMIT (STUDENTNAME, ASSIGNMENTNUM, SUBMITTIME) values (?,?,?)"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, u)
                ibm_db.bind_param(stmt, 2, x)
                ibm_db.bind_param(stmt, 3, t)
                ibm_db.execute(stmt)
            else:
                sql = "UPDATE SUBMIT SET SUBMITTIME = ? WHERE STUDENTNAME = ? and ASSIGNMENTNUM = ?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, t)
                ibm_db.bind_param(stmt, 2, u)
                ibm_db.bind_param(stmt, 3, x)
                ibm_db.execute(stmt)


            return render_template("studentsubmit.html", msg=msg, datetime=subtime, marks=ma)
            continue
    return render_template("studentsubmit.html", datetime=subtime, Marks=ma)
```

3.11

Rendering Facultyprofile.Html

```
35  @app.route("/facultyprofile")
36  def fprofile():
37      return render_template("facultyprofile.html")
```

3.12

Rendering Facutlystulist.Html

```
222 @app.route("/facultymarks")
223 def facultymarks():
224     data=[]
225     sql = "SELECT USERNAME from REGISTER WHERE ROLE=1"
226     stmt = ibm_db.prepare(conn, sql)
227     ibm_db.execute(stmt)
228     name = ibm_db.fetch_tuple(stmt)
229     while name!= False:
230         data.append(name)
231         name=ibm_db.fetch_tuple(stmt)
232     data1 = []
233     for i in range(0,len(data)):
234         y = data[i][0].strip()
235         data1.append(y)
236     data1 = set(data1)
237     data1 = list(data1)
238     print(data1)
239
240     return render_template("facultystulist.html", names = data1, le=len(data1))
```

Here, when a faculty clicks on the marks tab in the faculty profile html page we will get the student list.

3.13

Rendering Facultymarks.Html

```
@app.route("/marksassign/<string:stdname>", methods=['POST', 'GET'])
def marksassign(stdname):
    global u
    global g
    global file
    da  = []

    COS_ENDPOINT = "Enter your COS Endpoint"
    COS_API_KEY_ID = "Enter your COS api key"
    COS_INSTANCE_CRN = "Enter your COS CRN ID"
    cos = ibm_boto3.client("s3",
                           ibm_api_key_id=COS_API_KEY_ID,
                           ibm_service_instance_id=COS_INSTANCE_CRN,
                           config=Config(signature_version="oauth"),
                           endpoint_url=COS_ENDPOINT)
    output = cos.list_objects(Bucket="studentassignmentsb")
    output
    l=[]
    for i in range(0,len(output['Contents'])):
        j = output['Contents'][i]['Key']
        l.append(j)
    l
    u = stdname
    print(len(u))
    print(len(l))
    n = []
    for i in range(0,len(l)):
        for j in range(0,len(u)):
            if u[j]==l[i][j]:
                n.append(l[i])
```

```
    file = set(n)
    file = list(file)
    print(file)
    print(len(file))
    g = len(file)
    sql = "SELECT SUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt , 1, u)
    ibm_db.execute(stmt)
    m = ibm_db.fetch_tuple(stmt)
    while m != False:
        da.append(m[0])
        m = ibm_db.fetch_tuple(stmt)

    print(da)
    return render_template("facultymarks.html", file=file, g=g, marks=0, datetime=da)
```

Here we will be getting the list of assignments submitted by the student along with the files from object storage.

When the faculty checks the file and gives marks in the marks tab and clicks on that tick button, the marks entered need to be stored in the database. To do that.

```
@app.route("/marksupdate/<string:anum>", methods=['POST', 'GET'])
def marksupdate(anum):
    ma = []
    da = []
    mark = request.form['mark']
    print(mark)
    print(u)
    sql = "UPDATE SUBMIT SET MARKS = ? WHERE STUDENTNAME = ? and ASSIGNMENTNUM = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, mark)
    ibm_db.bind_param(stmt, 2, u)
    ibm_db.bind_param(stmt, 3, anum)
    ibm_db.execute(stmt)
    msg = "MARKS UPDATED"
    sql = "SELECT MARKS, SUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt , 1, u)
    ibm_db.execute(stmt)
    m = ibm_db.fetch_tuple(stmt)
    while m != False:
        ma.append(m[0])
        da.append(m[1])
        m = ibm_db.fetch_tuple(stmt)
        #ma = ma[0]
    print(ma)
    print(da)
    return render_template("facultymarks.html", msg =msg, marks = ma, g=g, file=file, datetime=da)
```

3.14

Logout

When anyone clicks on the logout button then it will be redirected to the index page.

```python
@app.route("/logout")
def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return render_template("logout.html")
```

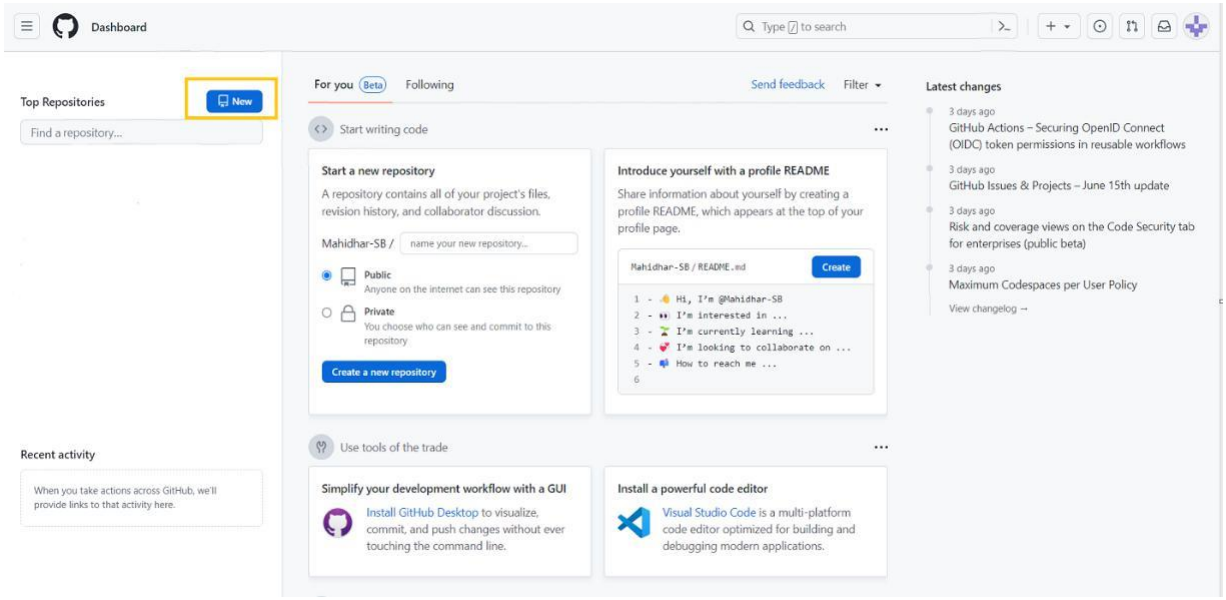
Here we will remove the saved email id and username in session, because now the user is not active so, no need to track.

4. GitHub

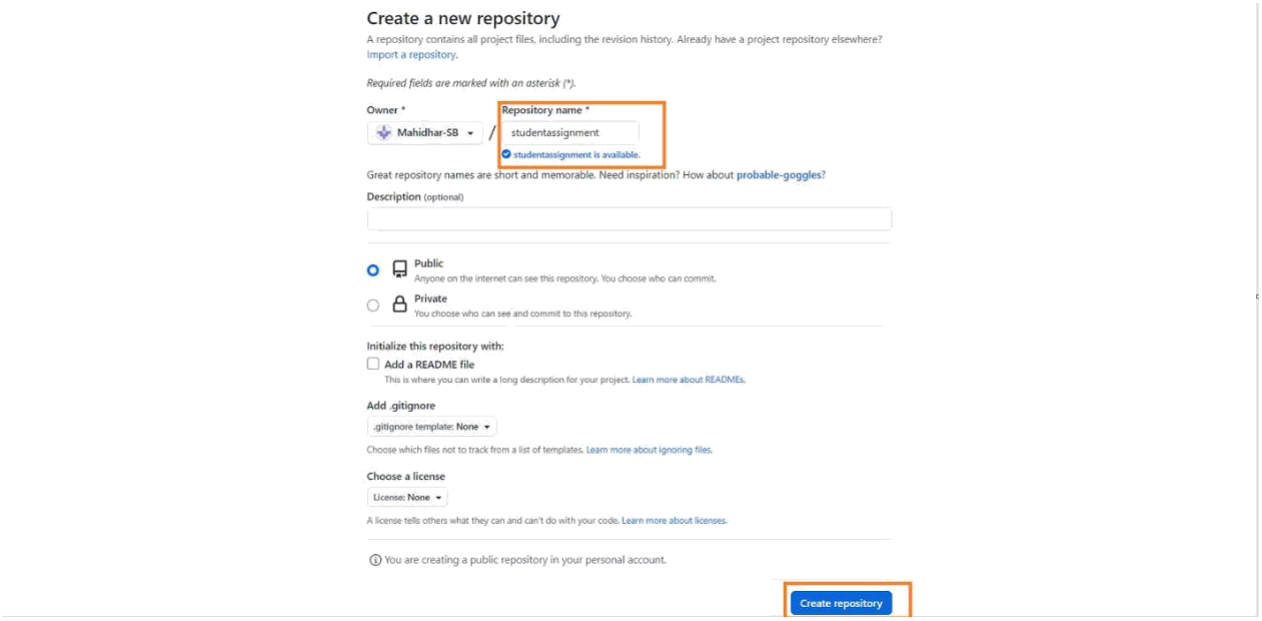
GitHub, Inc. is an Internet hosting service for software development and version control using Git. It provides the distributed version control of Git plus access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project.

4.1 Pushing All Files to GitHub Repository

First go to github.com, if you have an account sign in into it, else create an account in it. Once you login, you will be in the page some like this.

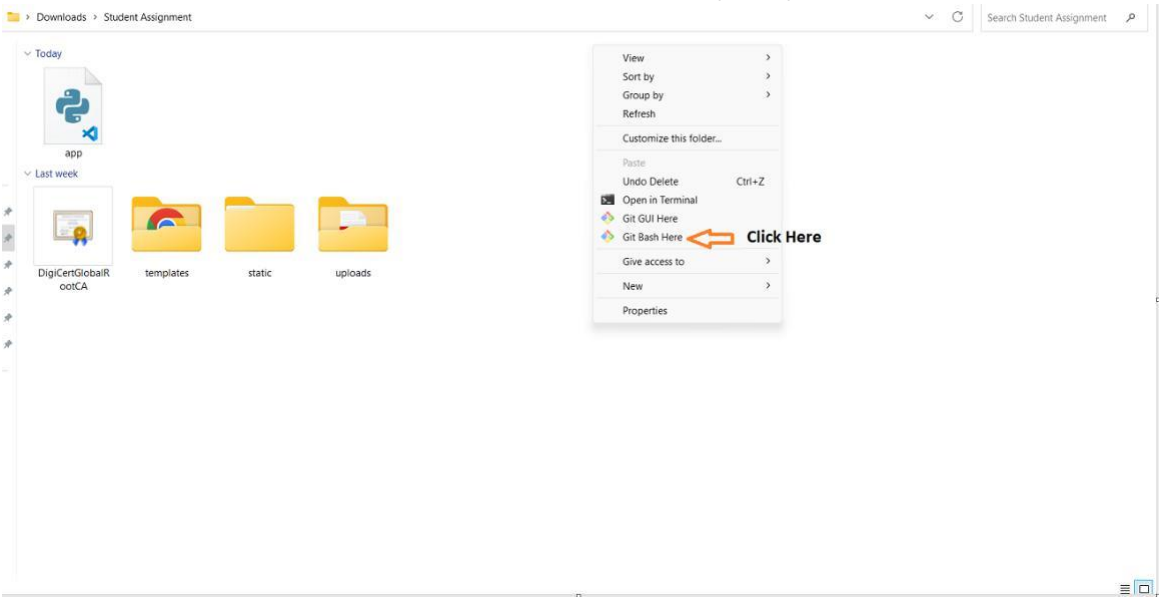


Now click on that new button to create a new repository. Where you can store all your files.



Now give the name for the repository. In that only we will store all our files. If you have available for name, you can scroll down and click on create repository. Now it will create a repository.

Now you have to download git - it's a version control software, where we can track the changes that have been made in the files. To download it click on this link. Select your system, download it and install it.



Go to your Folder in your file manager ? Give Right click ? show more properties (if windows 11) ? then click on Git Bash Here to open the git terminal.

Activity 1.1: Git Initialize

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment
$ git init
Initialized empty Git repository in C:/Users/smart/Downloads/Student Assignment
/.git/
```

Here we Initialise the git to track / start the version control of the project.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DigiCertGlobalRootCA.crt
        app.py
        static/
        templates/
        uploads/

nothing added to commit but untracked files present (use "git add" to track)
```

As you can see, all the files we have are untracked because we did make any changes after initialising the git. If you make that specific file will change to tracked.

Activity 1.2: Add all files to the staging area.
To add all files we will use git add -A

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git add -A
```

Once you enter this command, all files will be added to the staging area of git.

To check we use git status command

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   DigiCertGlobalRootCA.crt
        new file:   app.py
        new file:   static/images/author.jpg
        new file:   static/images/become.jpg
        new file:   static/images/blackboard.svg
        new file:   static/images/books.svg
        new file:   static/images/comment_1.jpg
        new file:   static/images/comment_2.jpg
        new file:   static/images/contact_background.jpg
        new file:   static/images/course_1.jpg
        new file:   static/images/course_2.jpg
        new file:   static/images/course_3.jpg
        new file:   static/images/course_4.jpg
        new file:   static/images/course_5.jpg
        new file:   static/images/course_6.jpg
        new file:   static/images/course_7.jpg
        new file:   static/images/course_8.jpg
        new file:   static/images/course_9.jpg
        new file:   static/images/courses_background.jpg
        new file:   static/images/earth-globe.svg
        new file:   static/images/envelope.svg
        new file:   static/images/event_1.jpg
```

As you can see, all the files in our folders have been added to the staging area.

Activity 1.3: Committing files to local git
Now whatever the files we have added, we will commit those to the local git. That means the files we have added are final, no changes needed, we can push them to the remote repository. To do this we will use git commit -m “Your message”.

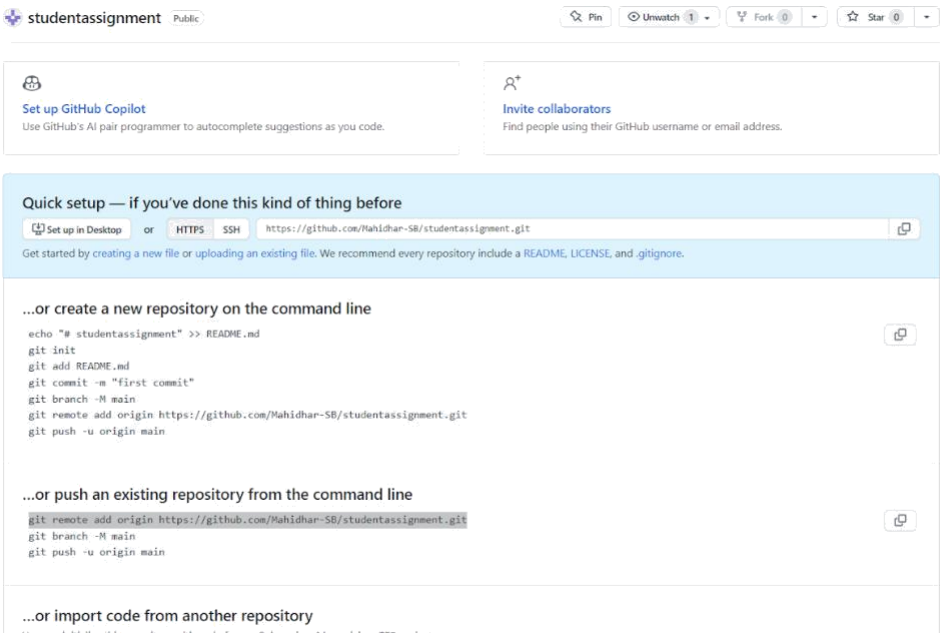
Note: -m means user defined message. While committing we will write a message for understanding of what we are committing.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git commit -m "Submitting all the files"
[master (root-commit) 3beca21] Submitting all the files
 128 files changed, 30499 insertions(+)
 create mode 100644 DigiCertGlobalRootCA.crt
 create mode 100644 app.py
 create mode 100644 static/images/author.jpg
 create mode 100644 static/images/become.jpg
 create mode 100644 static/images/blackboard.svg
 create mode 100644 static/images/books.svg
 create mode 100644 static/images/comment_1.jpg
 create mode 100644 static/images/comment_2.jpg
 create mode 100644 static/images/contact_background.jpg
 create mode 100644 static/images/course_1.jpg
 create mode 100644 static/images/course_2.jpg
 create mode 100644 static/images/course_3.jpg
```

Now if you check git status, it will show that all files are committed.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Activity 1.4: Add the remote git repo to local
The github repository we have created in github.com, we have added that to the git by using git remote add command.



We need the code that is highlighted in the grey colour.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git remote add origin https://github.com/Mahidhar-SB/studentassignment.git
```

Now once we add we can check. Adding has been successful or not by using git remote -v .

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git remote -v
origin  https://github.com/Mahidhar-SB/studentassignment.git (fetch)
origin  https://github.com/Mahidhar-SB/studentassignment.git (push)
```

Remote git has been added successfully.

Activity 1.5: Pushing files to github

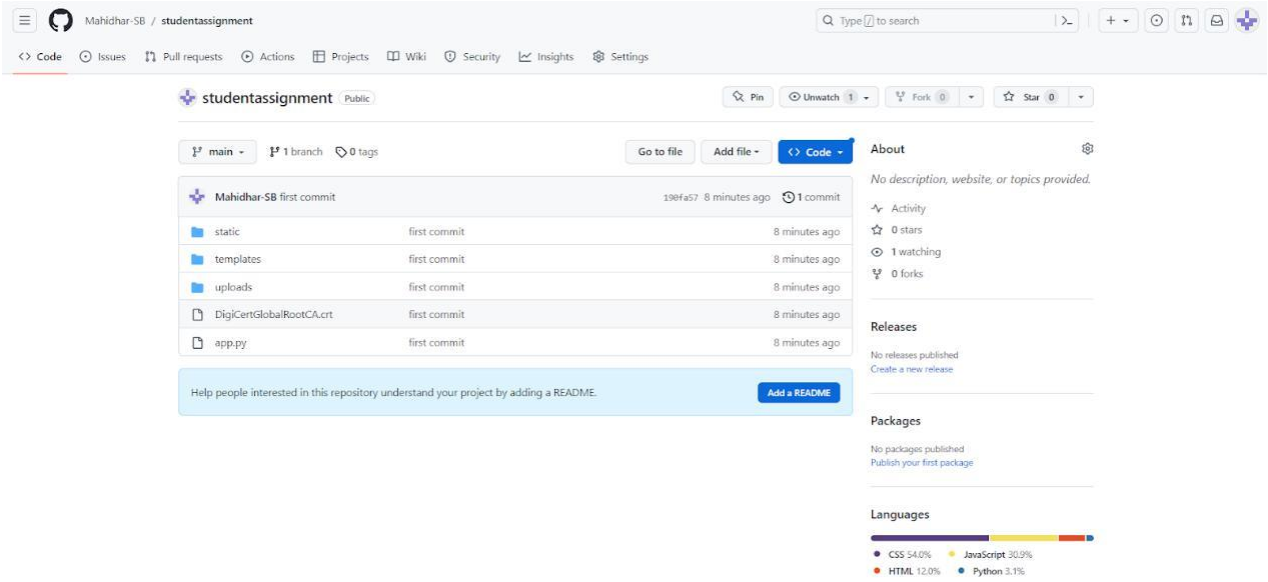
Now we need to push the files to a remote github repository. Before pushing we need to rename the master branch of local git to main by using git branch command.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (master)
$ git branch -m main
```

Here we are renaming the branch of the local system from master to main because in remote repo the branch is named as main.

```
smart@Mahidhar MINGW64 ~/Downloads/Student Assignment (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 141, done.
Counting objects: 100% (141/141), done.
Delta compression using up to 12 threads
Compressing objects: 100% (137/137), done.
Writing objects: 100% (141/141), 4.17 MiB | 2.06 MiB/s, done.
Total 141 (delta 22), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (22/22), done.
To https://github.com/Mahidhar-SB/studentassignment.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

We have successfully pushed all the files to github repo.



Containerize Your Application

Containerization is a method of packaging and deploying software applications in a way that allows them to run consistently across different environments. Containers are a lightweight alternative to virtual machines and are used to package and deploy applications as self-contained units.

The main advantage of containerization is that it allows for more efficient use of resources and greater flexibility in terms of deployment. Containers are isolated from one another, meaning that they can run on the same host without interfering with each other. This allows for multiple applications to be run on the same machine, which can save on hardware costs.

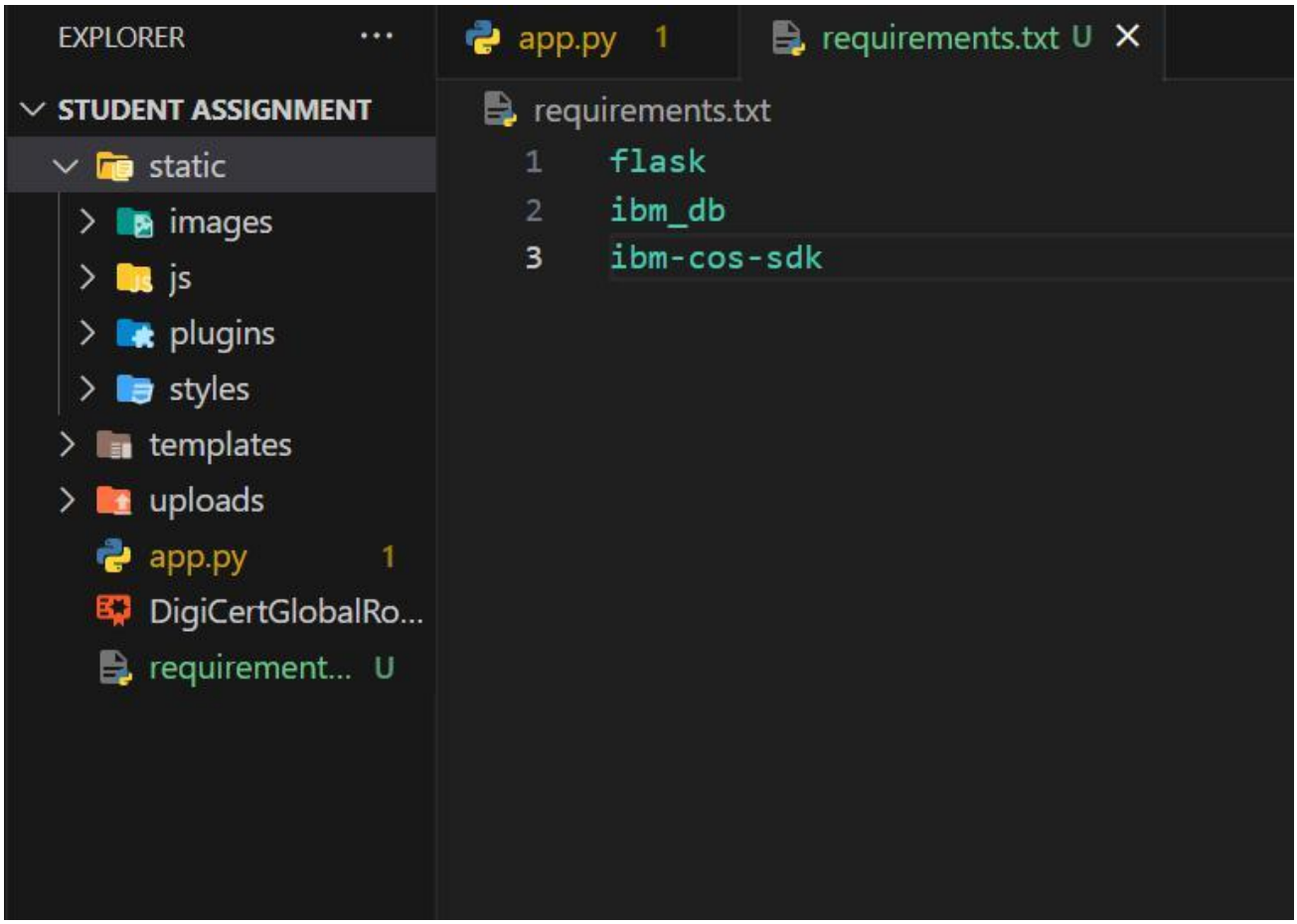
5.1

Create A Docker File

First to containerize your application, first we have to create a docker file. Where in that we will list all the instructions, commands and dependencies needed to run the application.

Activity 1.1: Creating requirements.txt file

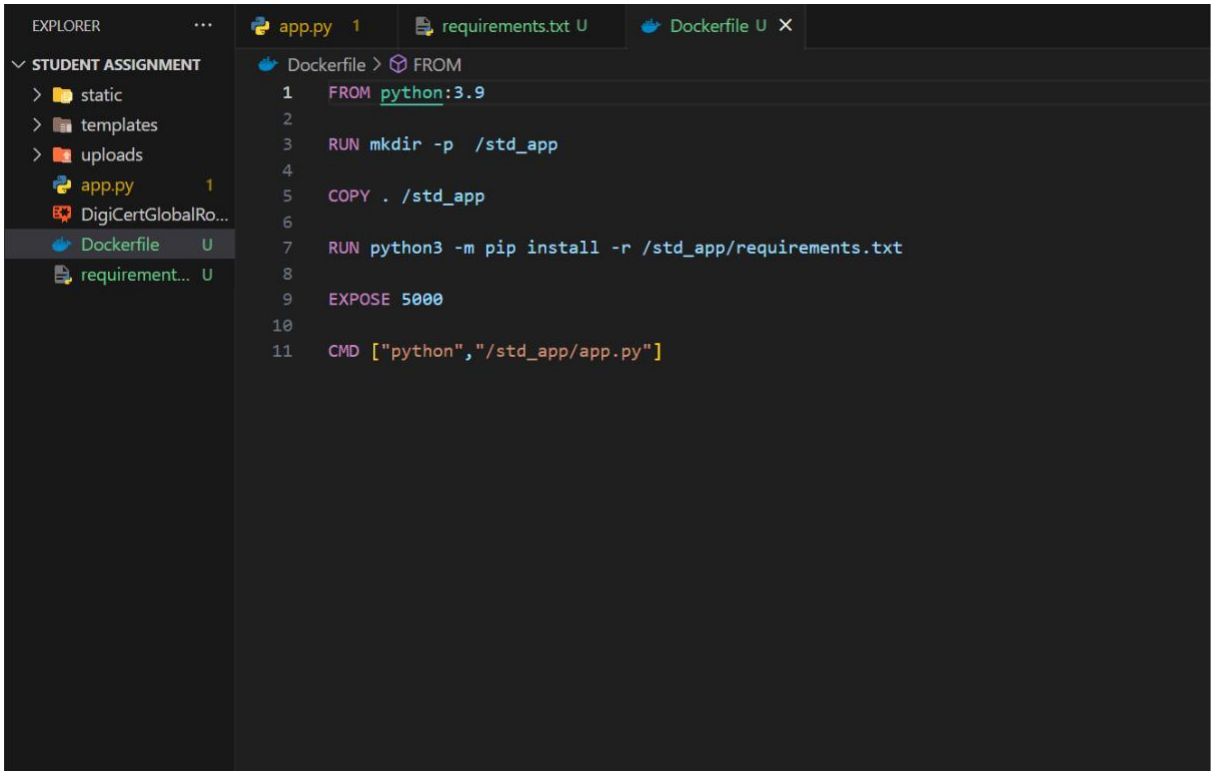
Here we will create or list out the requirements / dependencies of the application.



These are the list of libraries we need.

Activity 1.2: Creating Docker file

To Containerize your application using docker, you need to create a dockerfile in which you will have a list of instructions and commands.

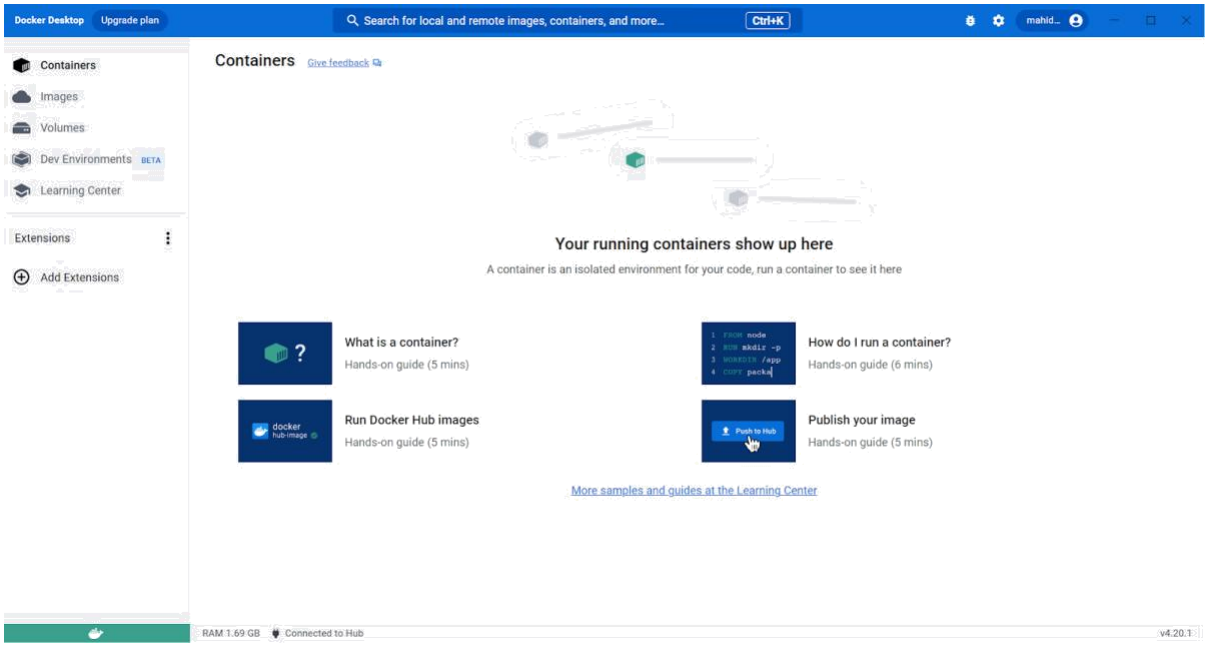


Activity 1.3: Build your docker image.

To create a docker image, we need to create a build / run the file we have created just now. To run it we need a docker desktop. So, download the docker desktop from this link. After downloading, install the docker desktop application and open it.

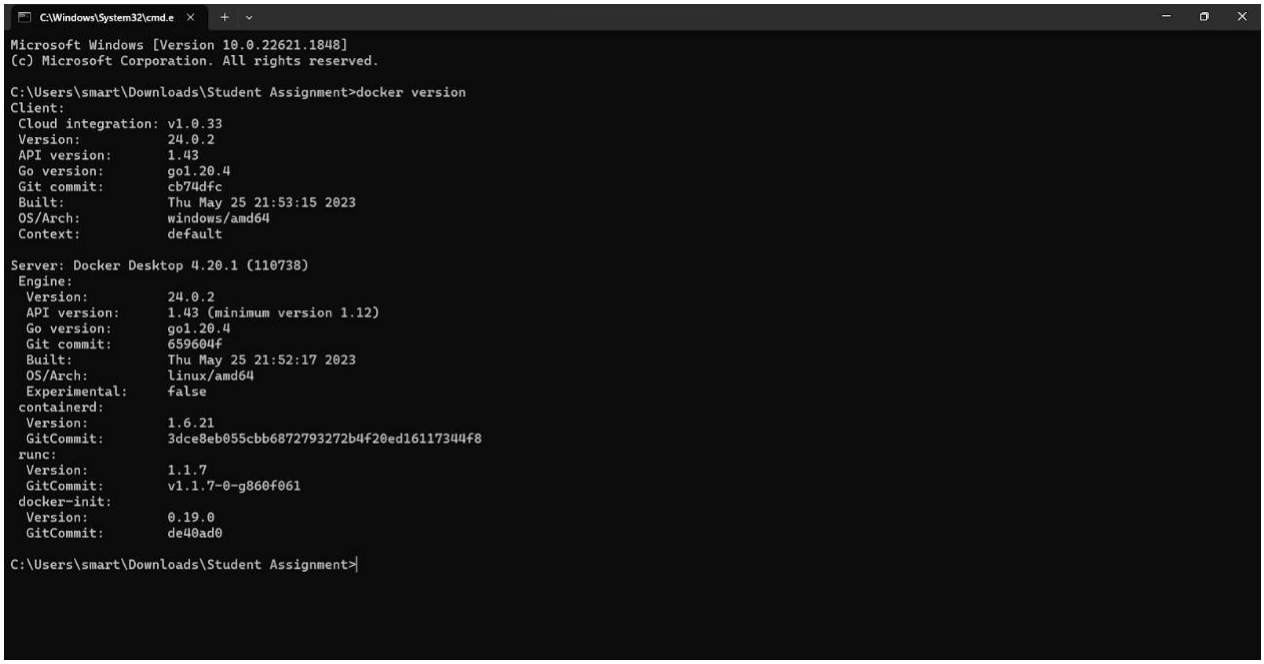
Note: when opening in windows if it asks for wsl. Download the wsl by using the link. File will be available in step 4, download and install it.

To get step by step guide to install docker desktop, you can go to this link



Now open the command prompt in the project folder.

To check if docker has been successfully installed or not, in the command prompt you can type for docker version.



If it returns the version and some details then installation is successful.

Now to create the application a image first, we have build the image

```
C:\Users\smart\Downloads\Student Assignment>docker build -t studentassignment:1.0 .
[+] Building 909.7s (10/10) FINISHED
 => [internal] load .dockerignore                                                                                0.1s
 => => transferring context: 2B                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                             0.1s
 => => transferring dockerfile: 209B                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.9                                                    5.5s
 => [auth] library/python:pull token for registry-1.docker.io                                                    0.0s
 => [internal] load build context                                                                                1.3s
 => => transferring context: 11.08MB                                                                             1.3s
 => [1/4] FROM docker.io/library/python:3.9@sha256:98f018a1afd67f2e17a4abd5bfe09b998734ba7c1ee54780e7ed216f8b8095c3  193.8s
 => => resolve docker.io/library/python:3.9@sha256:98f018a1afd67f2e17a4abd5bfe09b998734ba7c1ee54780e7ed216f8b8095c3  0.0s
 => => sha256:98f018a1afd67f2e17a4abd5bfe09b998734ba7c1ee54780e7ed216f8b8095c3 1.86kB / 1.86kB                    0.0s
 => => sha256:8d6fcdcf73c918683b9933499461b75bd20b5beb08ee96758bed081feb5a2d2d 2.01kB / 2.01kB                    0.0s
 => => sha256:618db5c0a924c7db9c3779147b637b806bcf6b23d11adde5de1f70aa353c6807 7.51kB / 7.51kB                    0.0s
 => => sha256:284f2345db055020282f6e80a646f1111fb2d5dfc6f7ee871f89bc50919a51bf 64.11MB / 64.11MB                142.8s
 => => sha256:bba7bb10d5baebcaad1d68ab3cbfd37390c646b2a688529b1d118a47991116f4 49.55MB / 49.55MB                131.7s
 => => sha256:ec2b820b8e87758dde67c29b25d4cbf88377601a4355cc5d556a9beebc80da00 24.03MB / 24.03MB                 32.6s
 => => sha256:fea23129f080a6e28ebff8124f9dc585b412b1a358bba566802e5441d2667639 211.00MB / 211.00MB              183.9s
 => => sha256:7c62c924b8a6474ab5462996f6663e07a515fab7f3fcdd605cae690a64aa01c7 6.39MB / 6.39MB                  142.1s
 => => extracting sha256:bba7bb10d5baebcaad1d68ab3cbfd37390c646b2a688529b1d118a47991116f4                        4.2s
 => => extracting sha256:ec2b820b8e87758dde67c29b25d4cbf88377601a4355cc5d556a9beebc80da00                        0.8s
 => => sha256:b2210932934efec1c984a1fad98942bb3f4d73106610e8db5185fc5037f18409 15.82MB / 15.82MB                150.6s
 => => extracting sha256:284f2345db055020282f6e80a646f1111fb2d5dfc6f7ee871f89bc50919a51bf                        4.2s
 => => sha256:ee9c01829d92f672ccecca7eaf47c437ad8ed4ca4f62101f7d6d2081bf54231e 242B / 242B                       143.2s
 => => sha256:d6285f41f1b6dc5d19e09aff87fbf4700c8df6453fb2608328f9aa648b08972c 2.85MB / 2.85MB                   146.8s
 => => extracting sha256:fea23129f080a6e28ebff8124f9dc585b412b1a358bba566802e5441d2667639                        8.0s
 => => extracting sha256:7c62c924b8a6474ab5462996f6663e07a515fab7f3fcdd605cae690a64aa01c7                        0.3s
 => => extracting sha256:b2210932934efec1c984a1fad98942bb3f4d73106610e8db5185fc5037f18409                        0.6s
 => => extracting sha256:ee9c01829d92f672ccecca7eaf47c437ad8ed4ca4f62101f7d6d2081bf54231e                        0.0s
 => => extracting sha256:d6285f41f1b6dc5d19e09aff87fbf4700c8df6453fb2608328f9aa648b08972c                        0.3s
 => [2/4] RUN mkdir -p  /std_app                                                                                 0.9s
 => [3/4] COPY . /std_app                                                                                        0.1s
 => [4/4] RUN python3 -m pip install -r /std_app/requirements.txt                                              708.3s
 => exporting to image                                                                                           0.9s
 => => exporting layers                                                                                          0.9s
 => => writing image sha256:f0cfc2d9762acf7044cf2141f8cd5e9a7135d0b4775b1c4344ecb9d5cd5a4b09                     0.0s
 => => naming to docker.io/library/studentassignment:1.0                                                         0.0s
```

We have created our application as a docker image. Now we can check it by using docker images command

```
C:\Users\smart\Downloads\Student Assignment>docker images
REPOSITORY          TAG       IMAGE ID       CREATED       SIZE
studentassignment   1.0       f0cfc2d9762a   3 hours ago   1.2GB
```

Activity 1.4: Run the docker image

To the docker image we have created, now we need to run that docker image. To run it we can use the docker run command.

```
C:\Users\smart\Downloads\Student Assignment>docker run -p 5000:5000 studentassignment:1.0
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 167-292-134
```

After getting there you can open any browser and type localhost:5000. Where it will open the project that is running by using the docker image.

Open a new tab in command prompt and type docker ps, it will list the current running docker images

```
PS C:\Users\smart> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                    NAMES
6837c88fa4f0   studentassignment:1.0  "python /std_app/app…"   12 seconds ago  Up 11 seconds  0.0.0.0:5000->5000/tcp   recursing_shtern
```

Activity 1.5: pushing docker image to Docker hub

Docker Hub is a hosted repository service provided by Docker for finding and sharing container images with your team.

Note: To push the images to docker hub, firstly you need to create an account in the docker hub website. Click here, once you created the account sign in and now go to docker desktop and click on sign in the right top to sign in with the same account in the docker desktop too.

Before pushing first we need to tag the local image with the docker hub username/repository name/tag

```
C:\Users\smart\Downloads\Student Assignment>docker tag studentassignment:1.0 mahidhar23/studentassignment:1.0
```

Once tag is successful you can check it by using docker images, tag will create a copy of our original image with a new name.

Now just before push, you need to login in the command line too. So, to do that type docker login. It will take existing credentials and it will push.

```
C:\Users\smart\Downloads\Student Assignment>docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
```

Now we can push our image to docker hub by a simple docker command i.e docker push <tagged image name>

```
C:\Users\smart\Downloads\Student Assignment>docker push mahidhar23/studentassignment:1.0
The push refers to repository [docker.io/mahidhar23/studentassignment]
4973ad265ee7: Pushed
86733fa5145b: Pushed
f75db63f689b: Pushed
68361737f8ab: Pushed
d489e554906d: Pushed
c76ff3d38b1d: Pushed
037f26f86912: Pushed
e67fb4bad8f4: Pushed
964529c819bb: Pushed
2f98f42985b1: Pushed
332b199f36eb: Pushed
1.0: digest: sha256:8b40ca698d3cd66843e70f9e4ee0d3ed138f228b5e9824c0bf1aea9b1c1d5107 size: 2637
```

6.

Kubernetes

Kubernetes (often abbreviated as "K8s") is an open-source container orchestration system for automating the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation.

Kubernetes also has a rich ecosystem of tools and extensions that can be used to manage and monitor containerized applications, including monitoring, logging, and networking.

7

Creating YAML File

YAML (which stands for YAML Ain’t Markup Language) is a language used to provide configuration for software, and is the main type of input for Kubernetes configurations. It is human-readable and can be authored in any text editor.

deployment.yaml

C: > Users > smart > Downloads > deployment.yaml

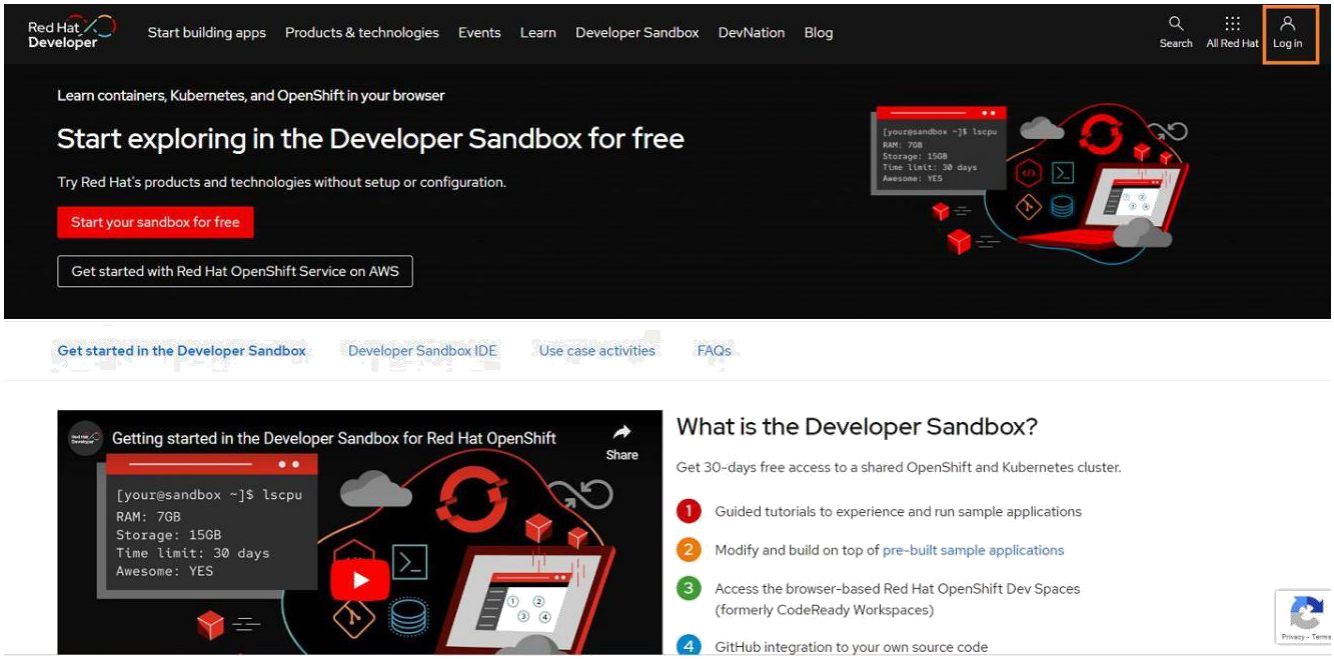
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: std-app-deploy
  labels:
    app: std-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: std-app
  template:
    metadata:
      labels:
        app: std-app
    spec:
      containers:
        - name: std-app
          image: mahidhar23/studentassignment:1.0
          ports:
          - containerPort: 5000

```

Here we have two yaml files, one file has configurations for deployment and the other is for creating service for the deployment.

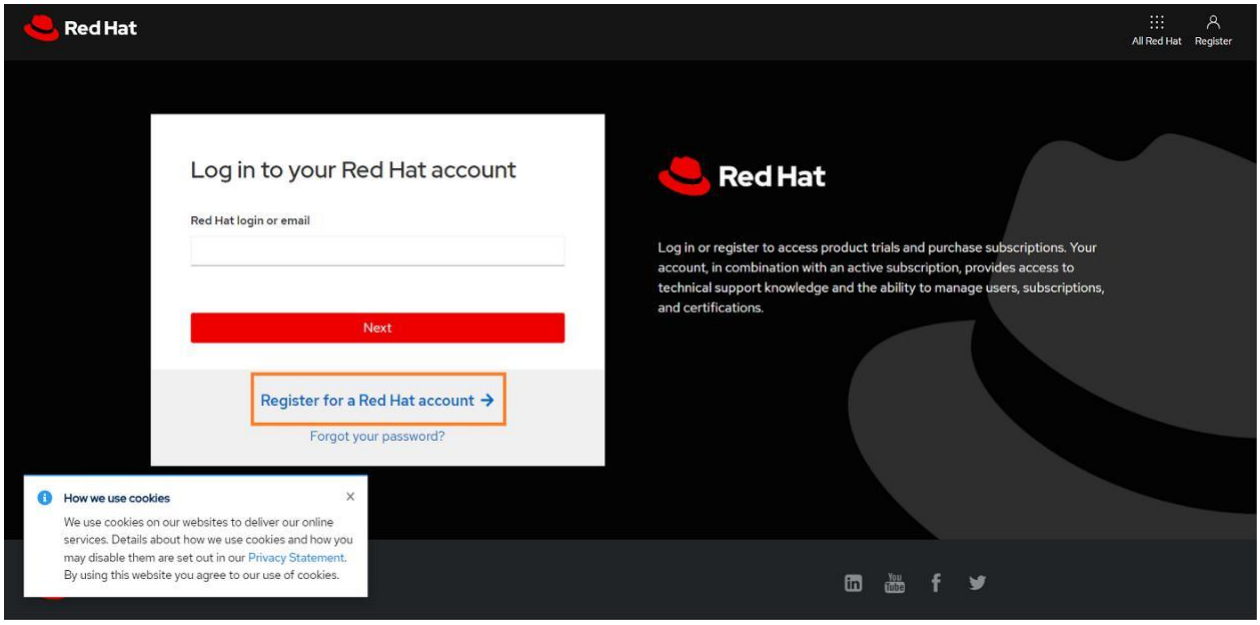
Deployment With Openshift

Red Hat OpenShift is a cloud-based Kubernetes platform that helps developers build applications. It offers automated installation, upgrades, and life cycle management throughout the container stack — the operating system, Kubernetes and cluster services, and applications — on any cloud.

First to use openshift you need to create a red hat account. To create it go to this url

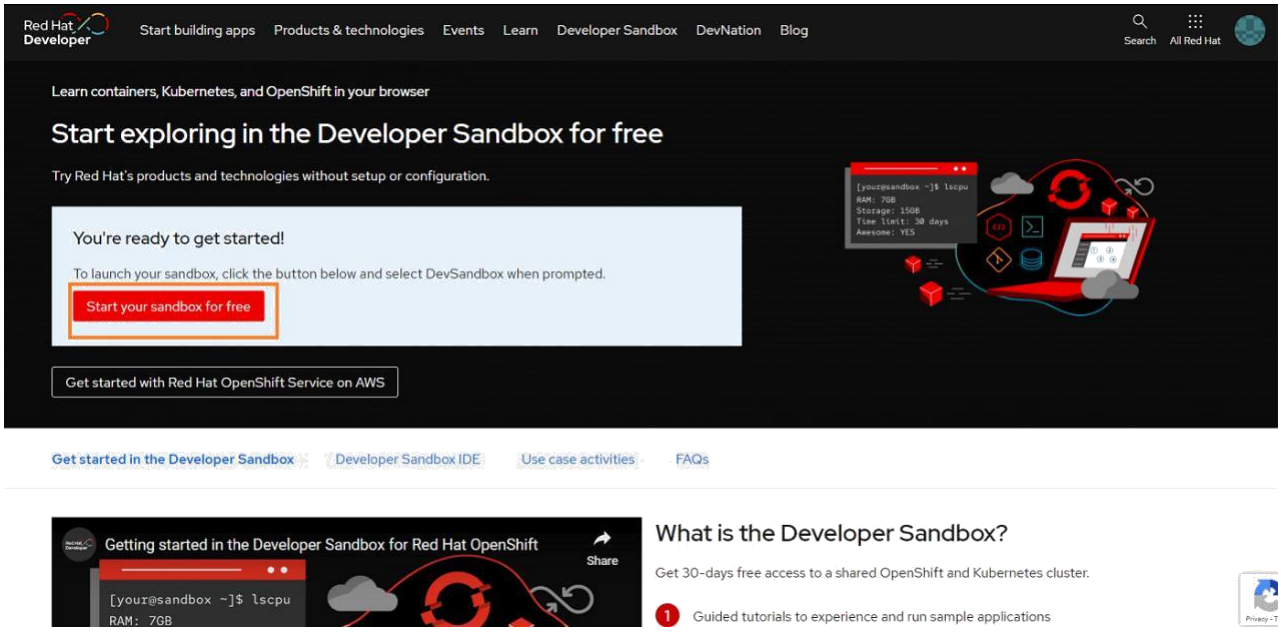


Click on the top right side login, you will be redirected to the login page.

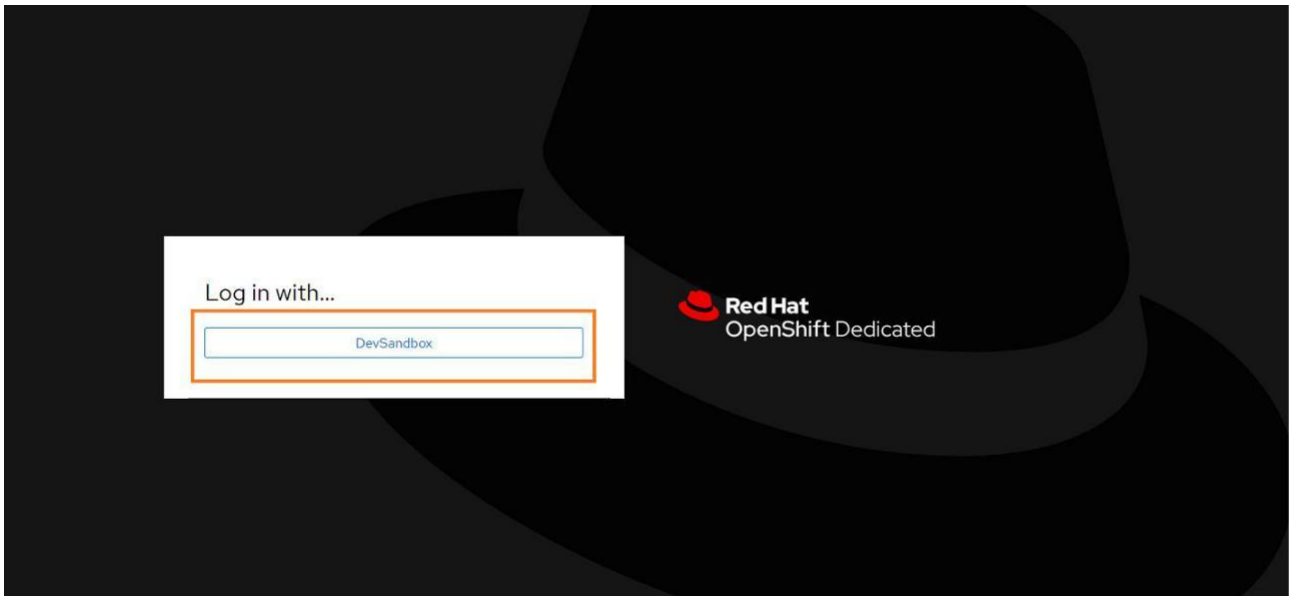


Now click on register for red hat account, now you will be redirected to a registration form. Fill in the necessary details. Once that is done you will be redirected to the main page, no login by giving the email and password which you gave just now.

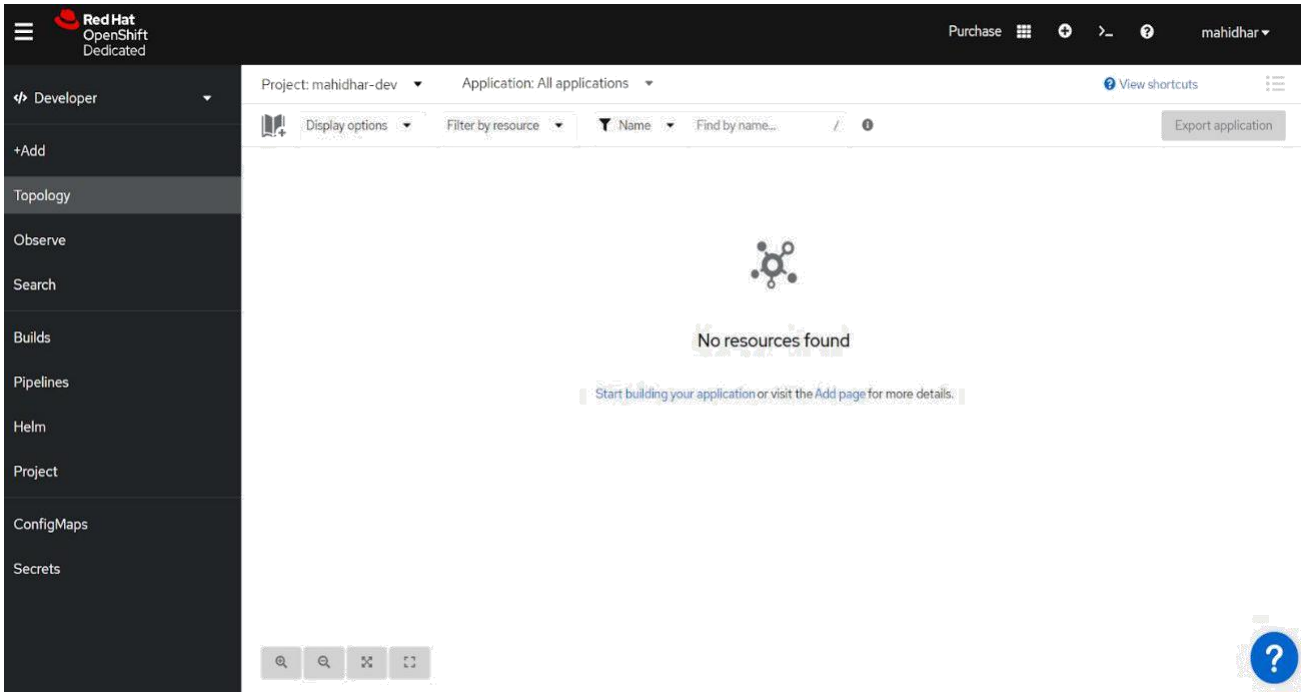
Once logged in, now click on start your sandbox for free..



Once you click on it will ask for login with a sandbox, there also you need to create the account, give the same details you have given above. Once it's done. It will ask for login with dev sandbox



Now click on it to login and once you set the developer role, it will bring you to the this page

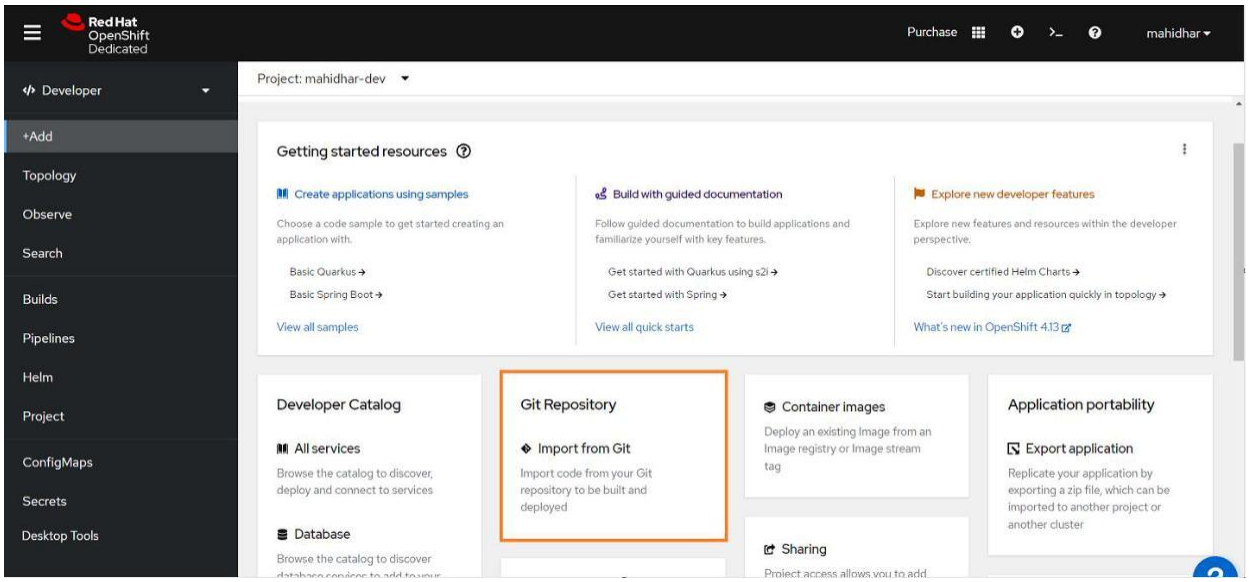


Stay on this page as developer and now click on that +add button on the left side panel. Here we can deploy our application in three ways.

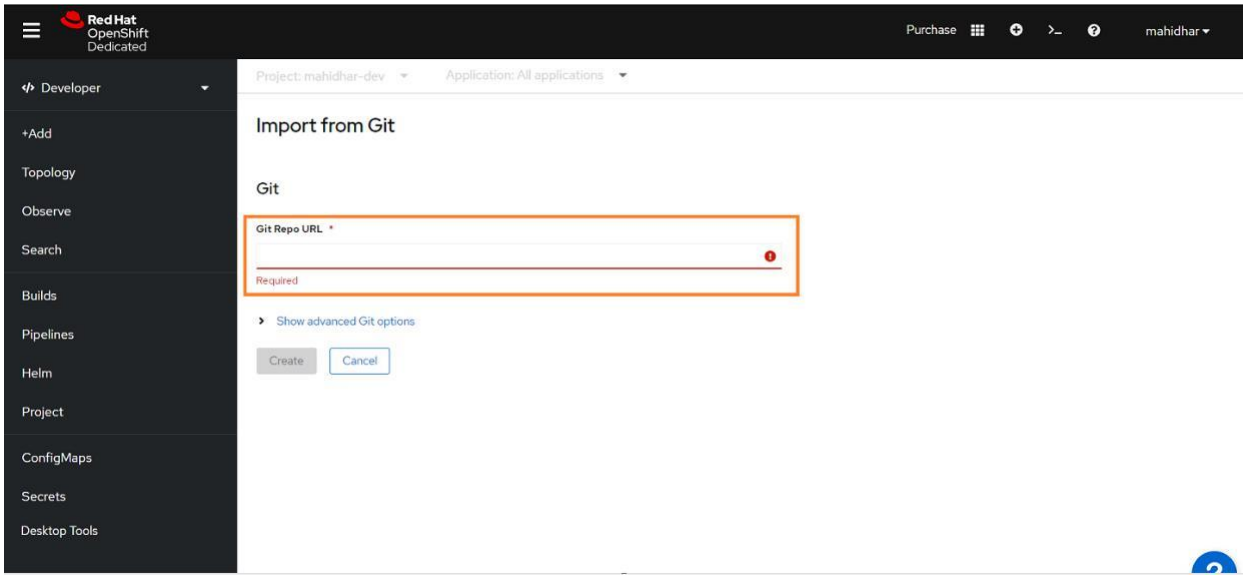
9.

Deploying Application In Openshift Using Github

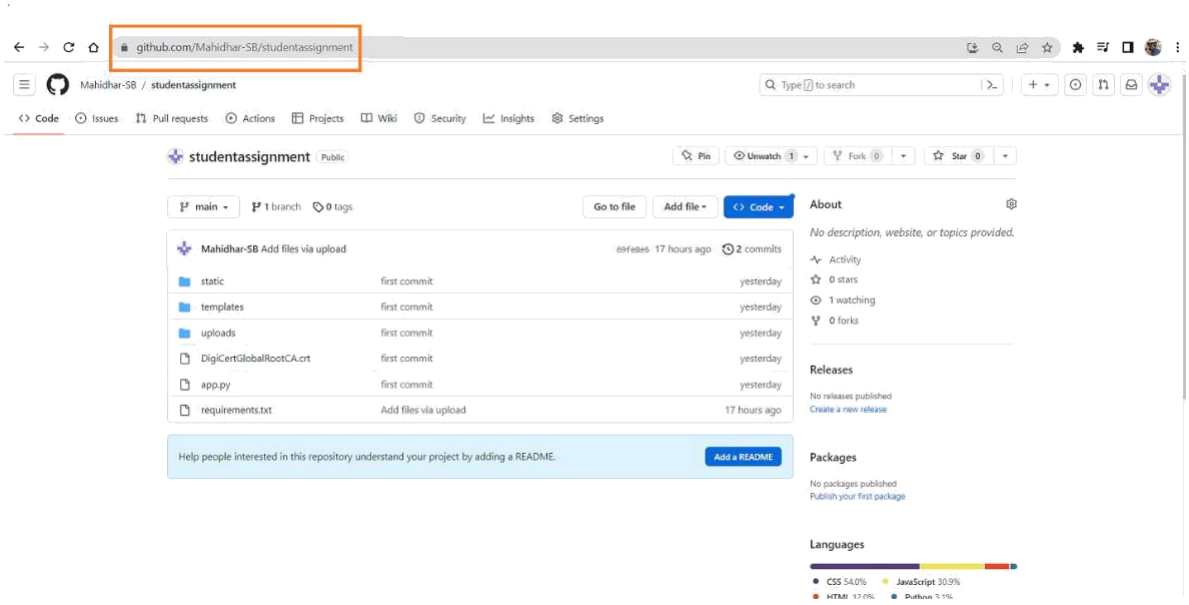
Click on that import from git option



After clicking on now in, give the github url of the git repo in which we have pushed our application.

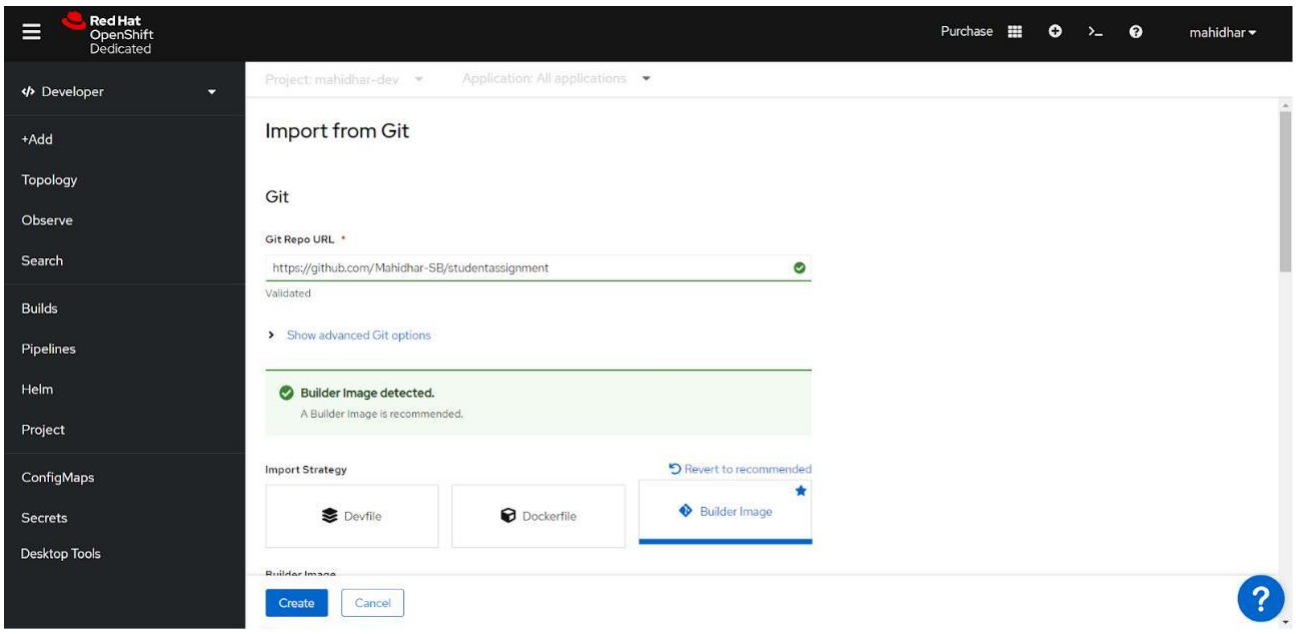


Go to github.com, login with your account and in repositories click on your project repository and copy the url of that repository.



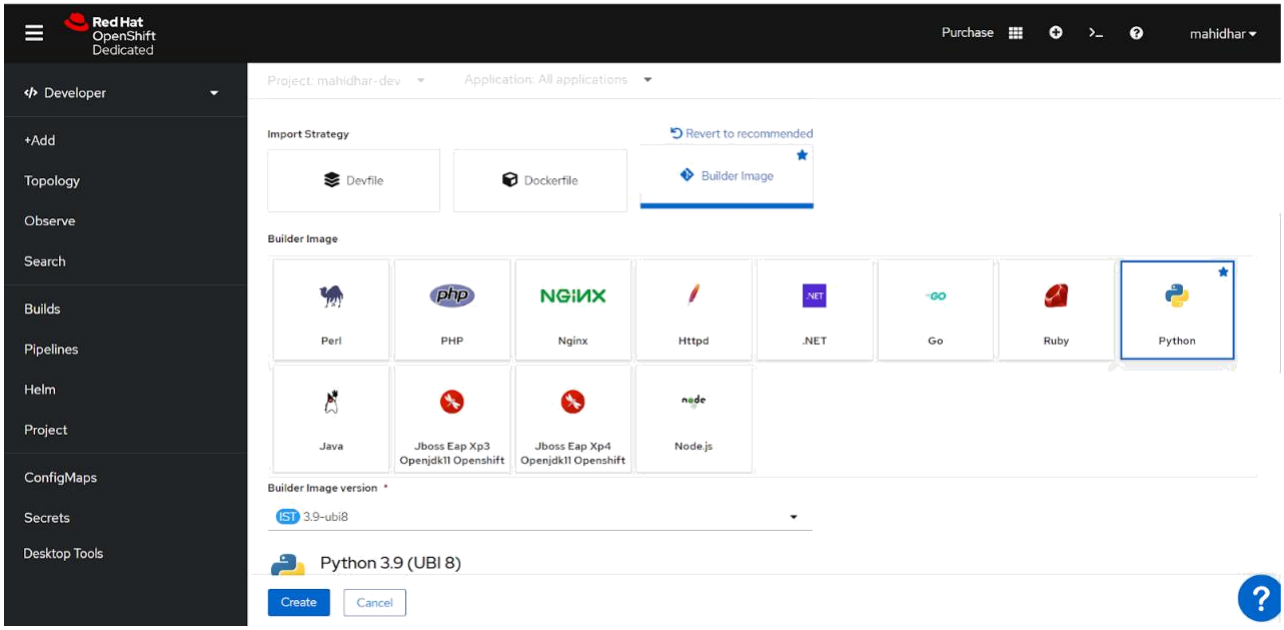
Note: You need to add the requirements.txt file. To add it, click on add file button (just next to that blue code button) ? now select upload files ? click on choose files ? choose the file in the pop up window the local project folder ? click on open in the pop up window ? no scroll down and click on commit button.

Once this done copy the url and paste in the openshift sandbox page

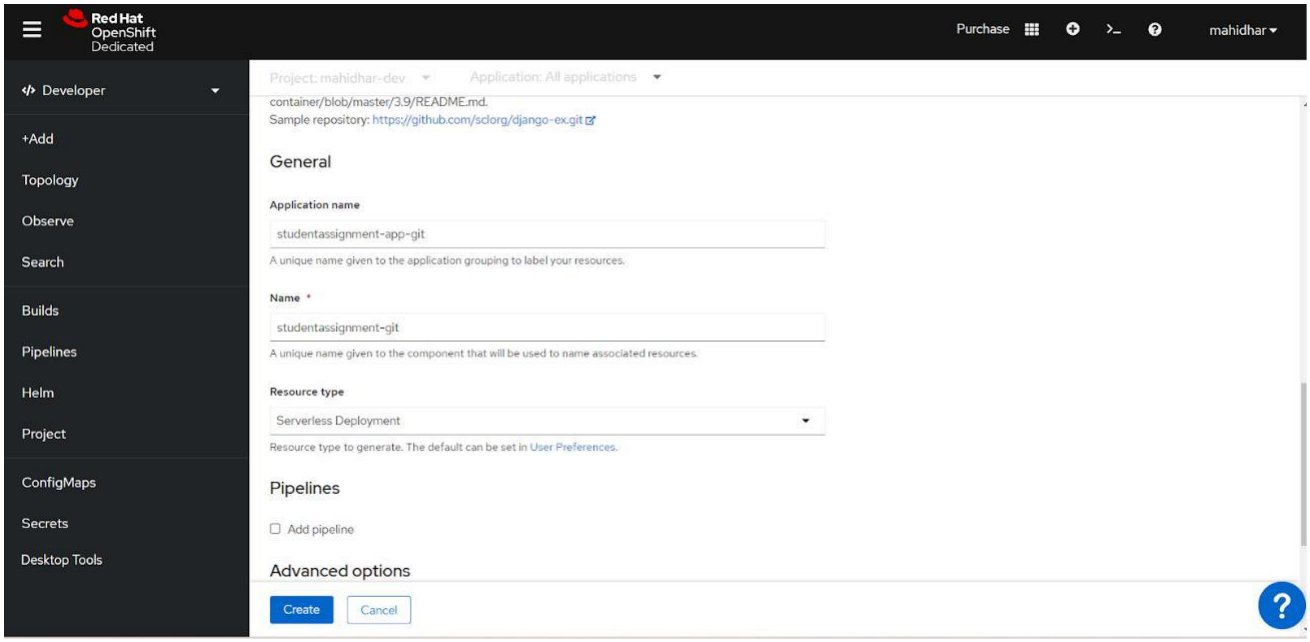


Once you paste it, validate and give us a green tick mark that path is correct. Now it will open a lot of options.

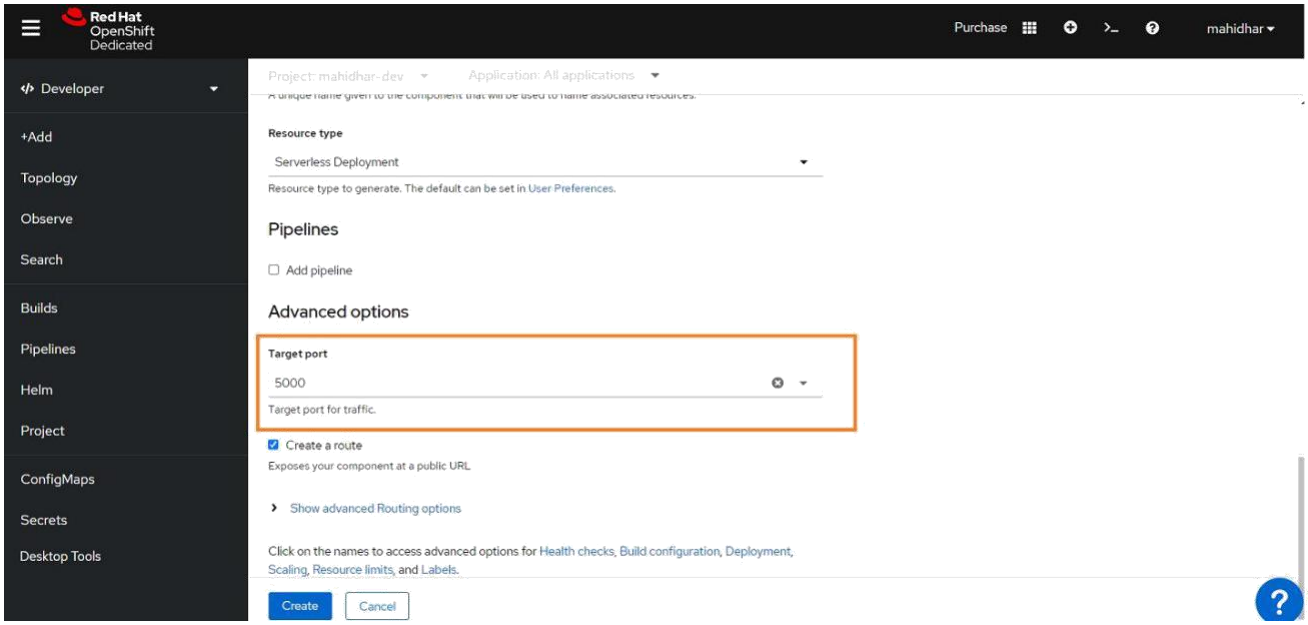
We need to select the builder image, by which programming language it has to run. To select it scroll a little bit down.



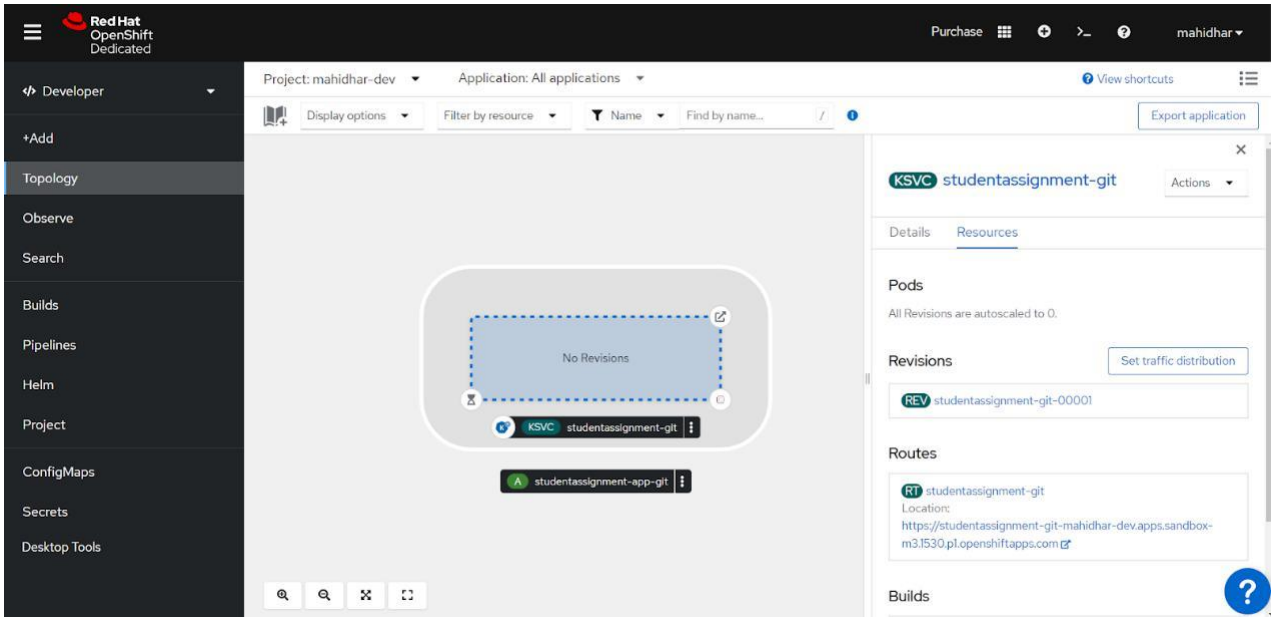
Now select python, as we used python programming language.
Now again scroll a little bit down, where in general section we can give some name for the application



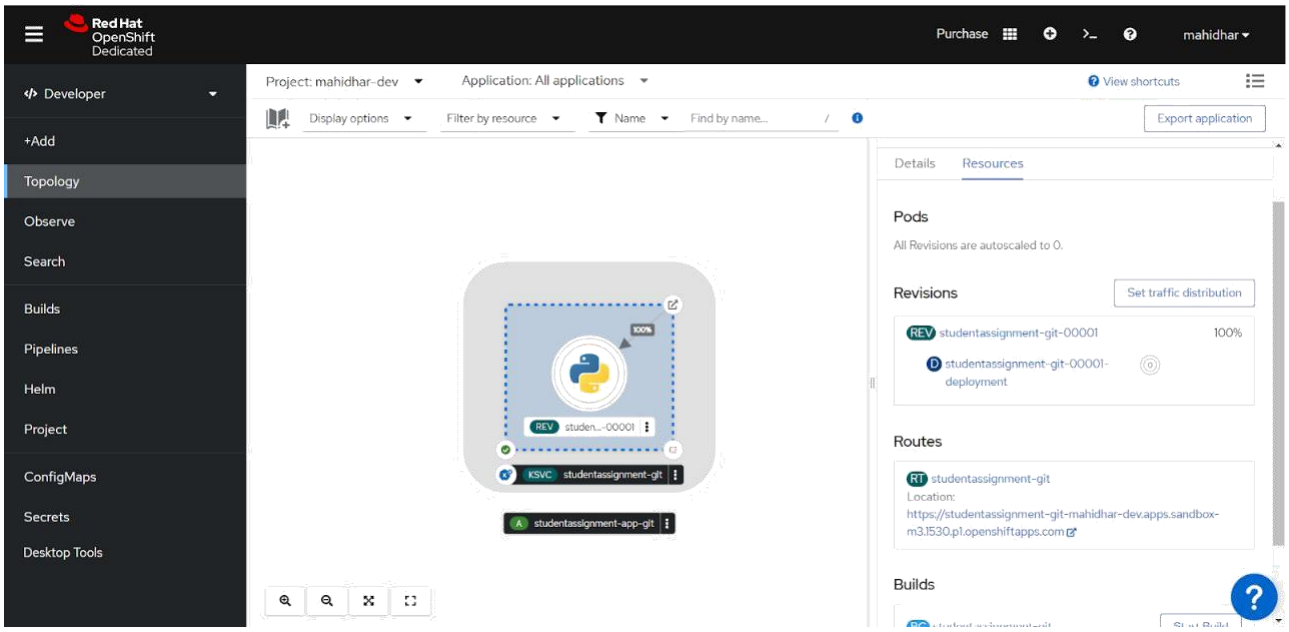
Now again scroll down a little bit in the advance option section where you will see the target port, in this we need to give the port address of our application. Which is default port address i.e 5000 (If you have specified port address in the flask app.run section, you need to give that port address instead of 5000).



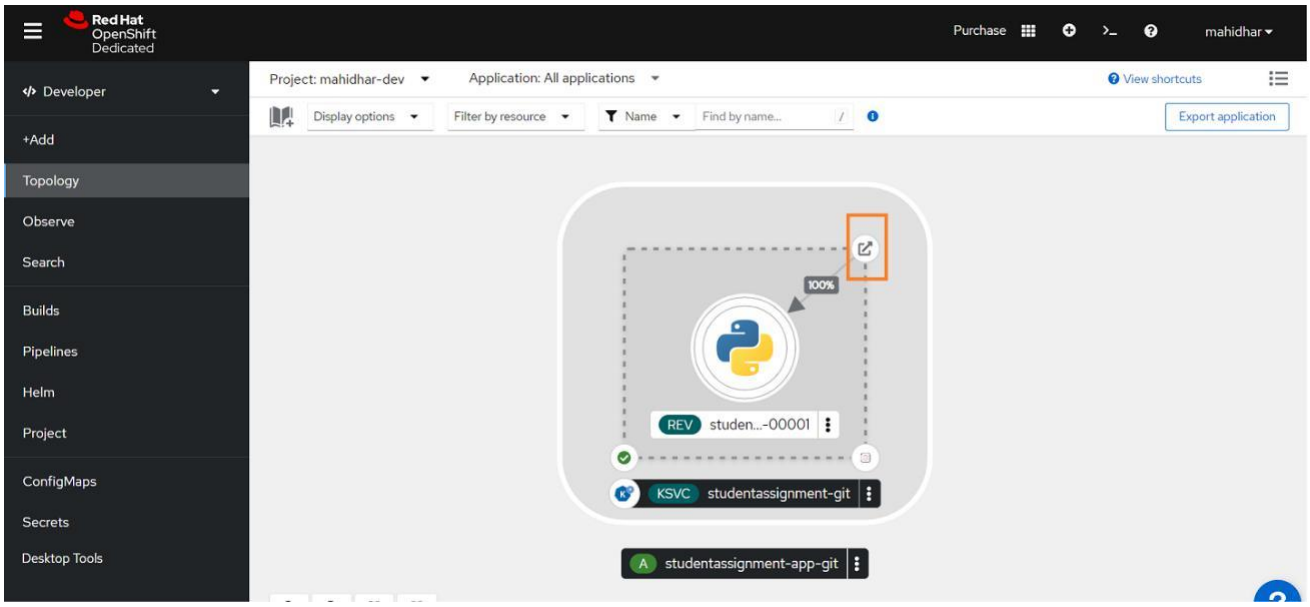
Once that is done click on create option. It will create the application and it will take some time to build the application because it has to build docker and kubernetes for the application. If you click on that rounded rectangle box, you will get the right side menu where you can see the status of the application building.

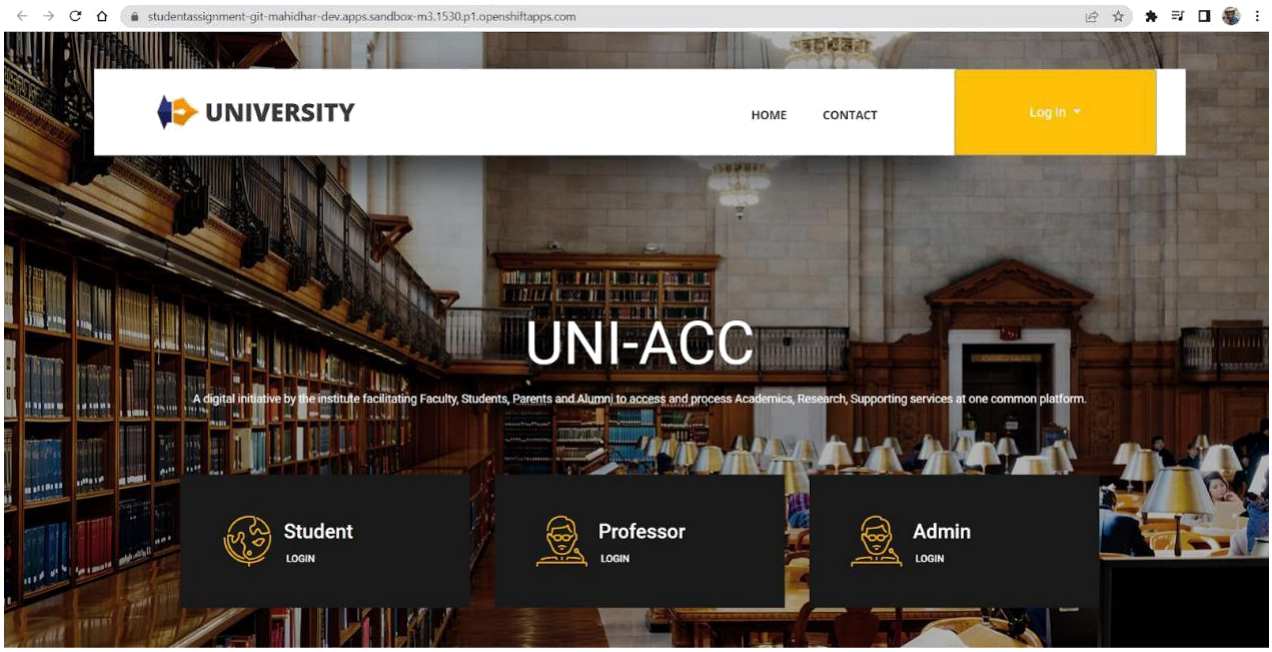


Once it's done, build the application the application will change like this.



Now you can click on this to publicly access your application.



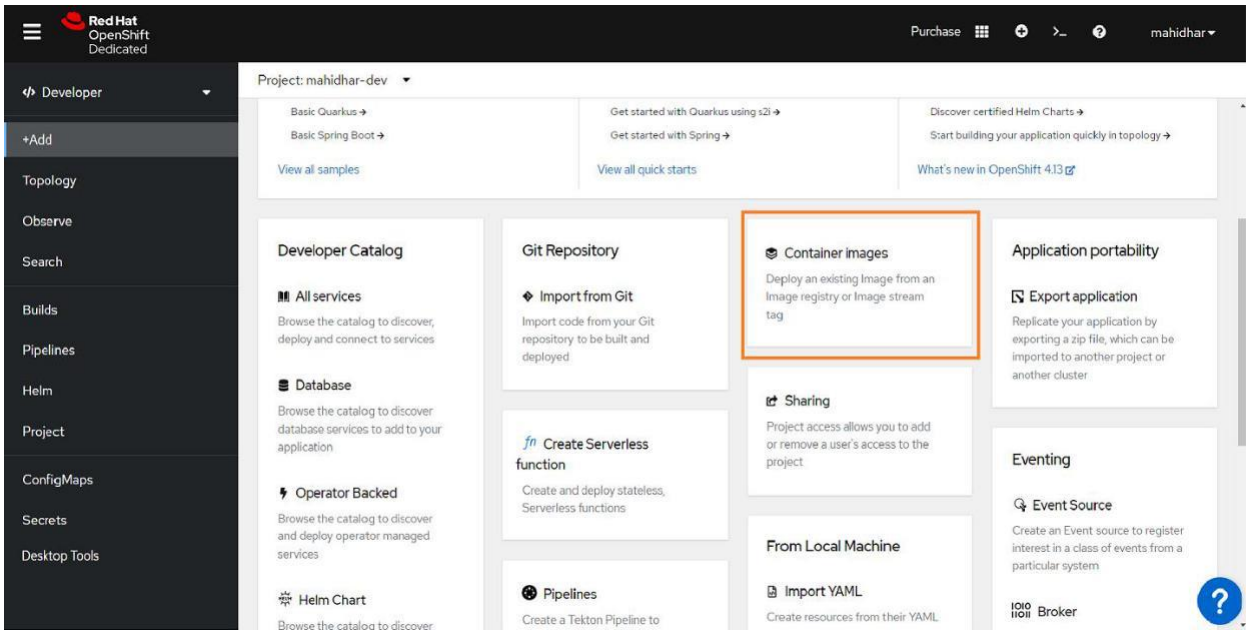


See the url section, the public access url will be created by the route service in the openshift.

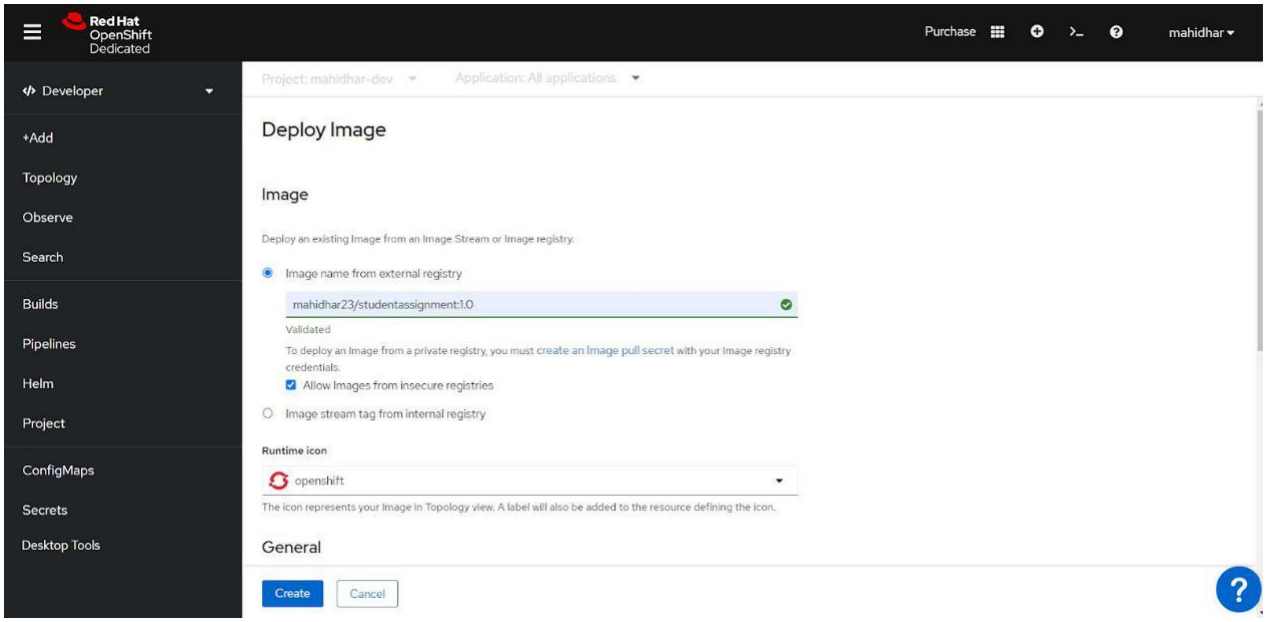
9.2

Deploying Application In Openshift Using Docker Image

Deploying the application using image is also the same as github, go to that add button on the left side panel and select the container images block.

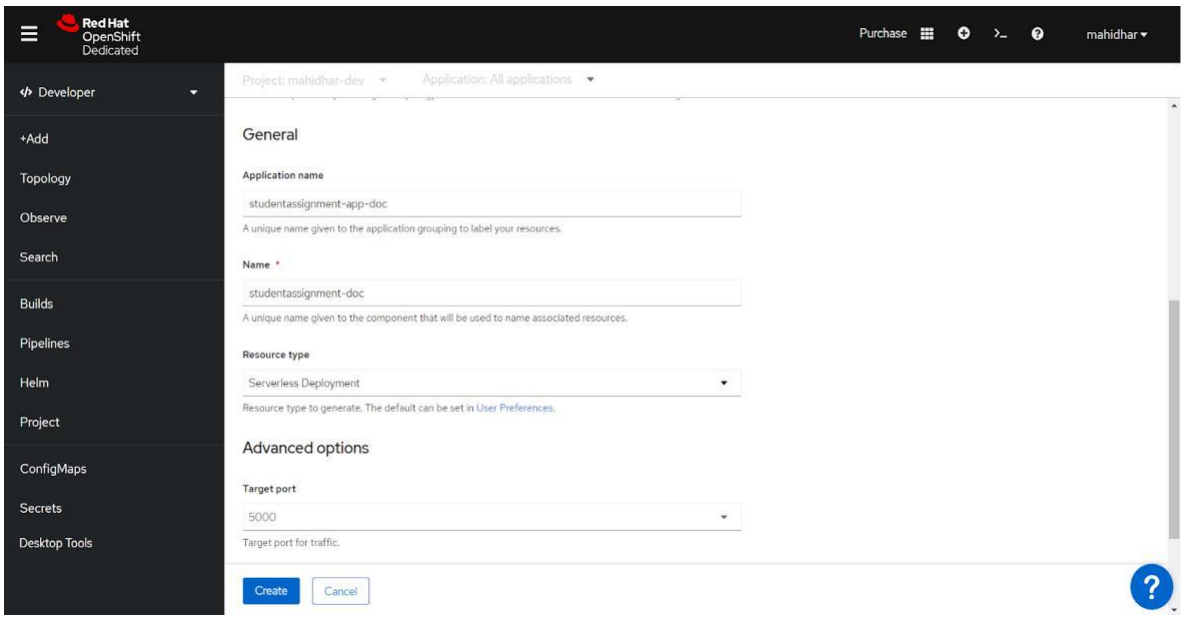


Once you click on it, it will take you to the new page where you have to give the docker image address that we pushed to docker hub.

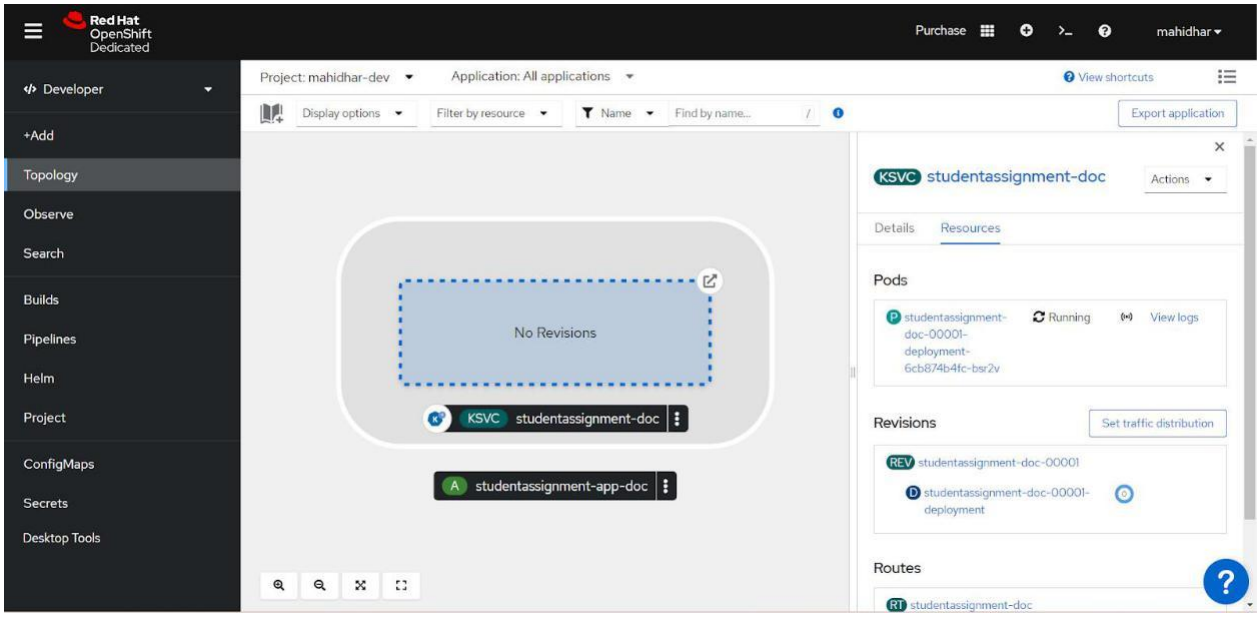


After giving the images address, it will validate and give us a green tick mark if the address is correct. After this make sure that allow images from insecure registries is ticked.

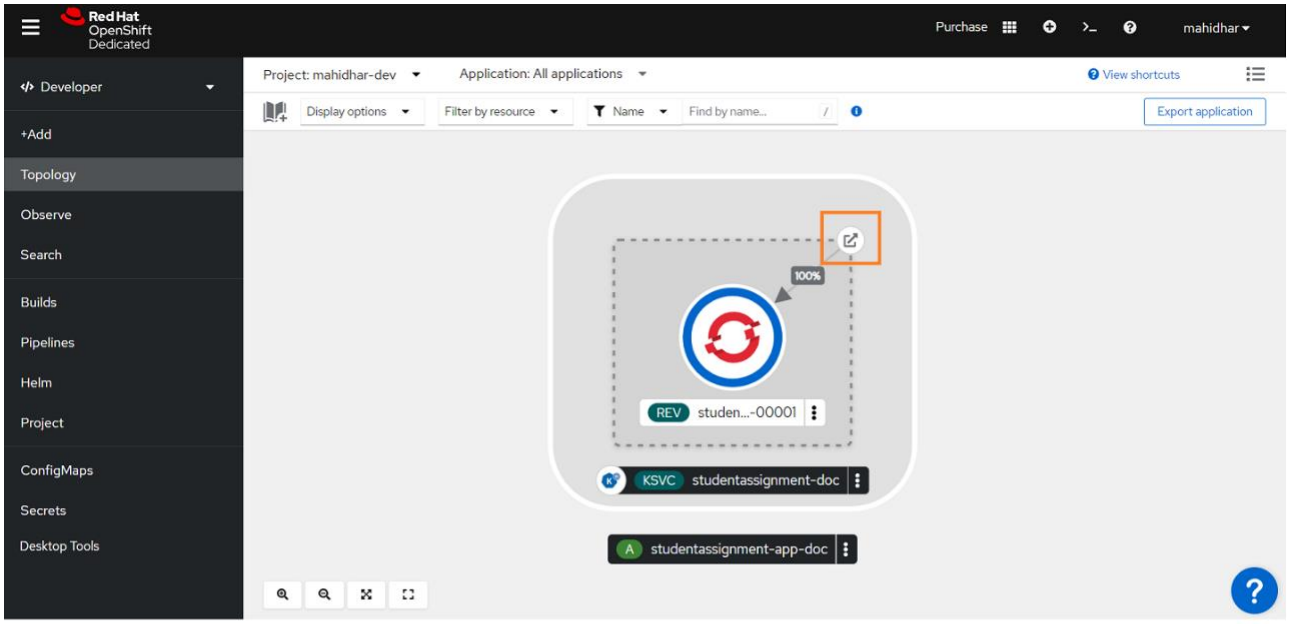
Now the same thing as we did above, name the application in the general section and port address as 5000 (or same port address of the flask) in the advanced options section.

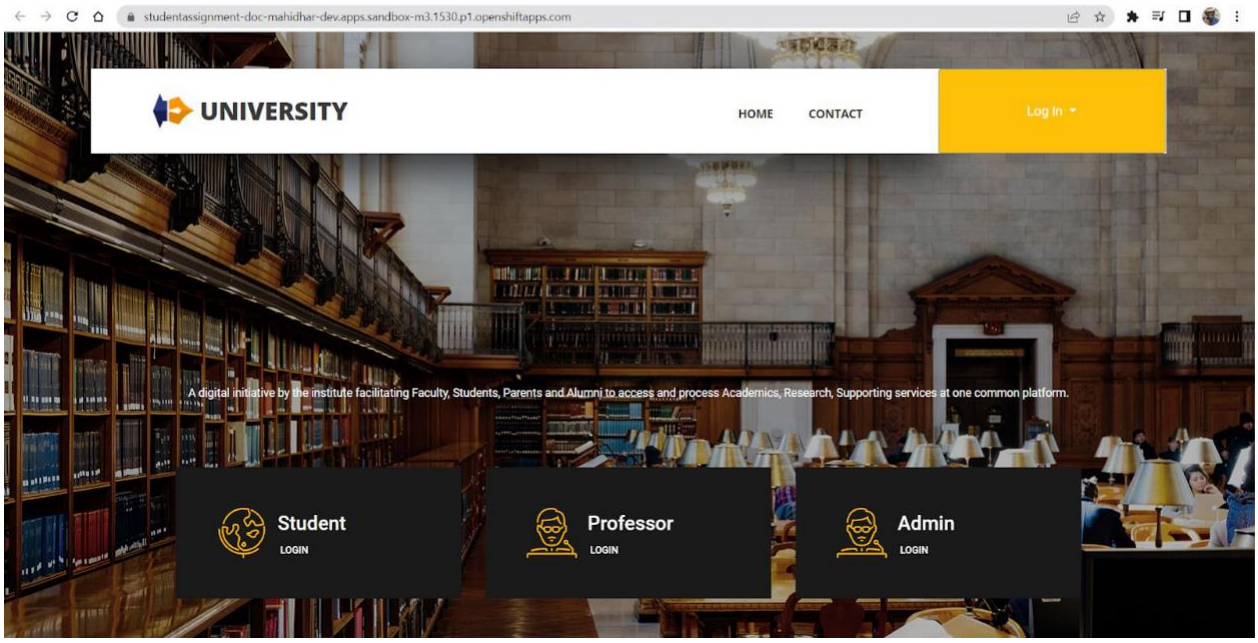


Once after giving all these, click on create. Same as above it will build the application, now based on the docker image.



It will take some time to build, once it is done the same as above the image of the application will change. Then click on the same open url button to open the application with a public url.

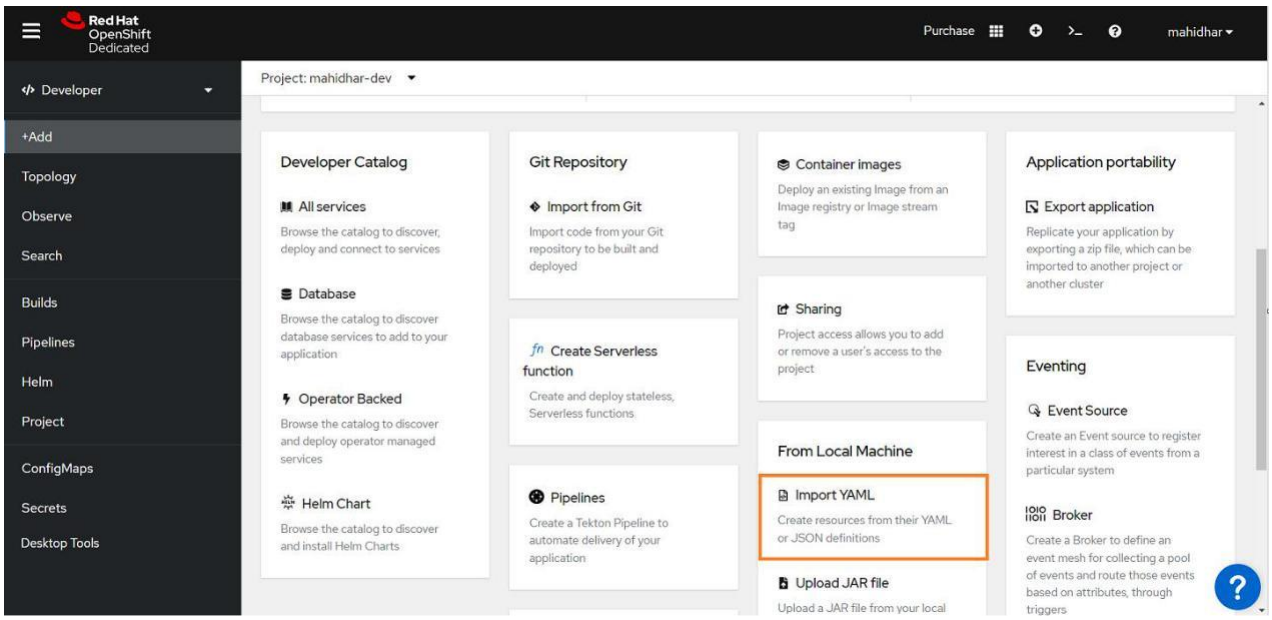

Red Hat
OpenShift
Dedicated
Purchase
mahidhar
Developer
+Add
Topology
Observe
Search
Builds
Pipelines
Helm
Project
ConfigMaps
Secrets
Desktop Tools
Project: mahidhar-dev
Application: All applications
View shortcuts
Display options
Filter by resource
Name
Find by name...
Export application
100%
REV
studen...-00001
KSVC
studentassignment-doc
A
studentassignment-app-doc


studentassignment-doc-mahidhar-dev.apps.sandbox-m3.1530.p1.openshiftapps.com
UNIVERSITY
HOME
CONTACT
Log In
A digital initiative by the institute facilitating Faculty, Students, Parents and Alumni to access and process Academics, Research, Supporting services at one common platform.
Student
LOGIN
Professor
LOGIN
Admin
LOGIN

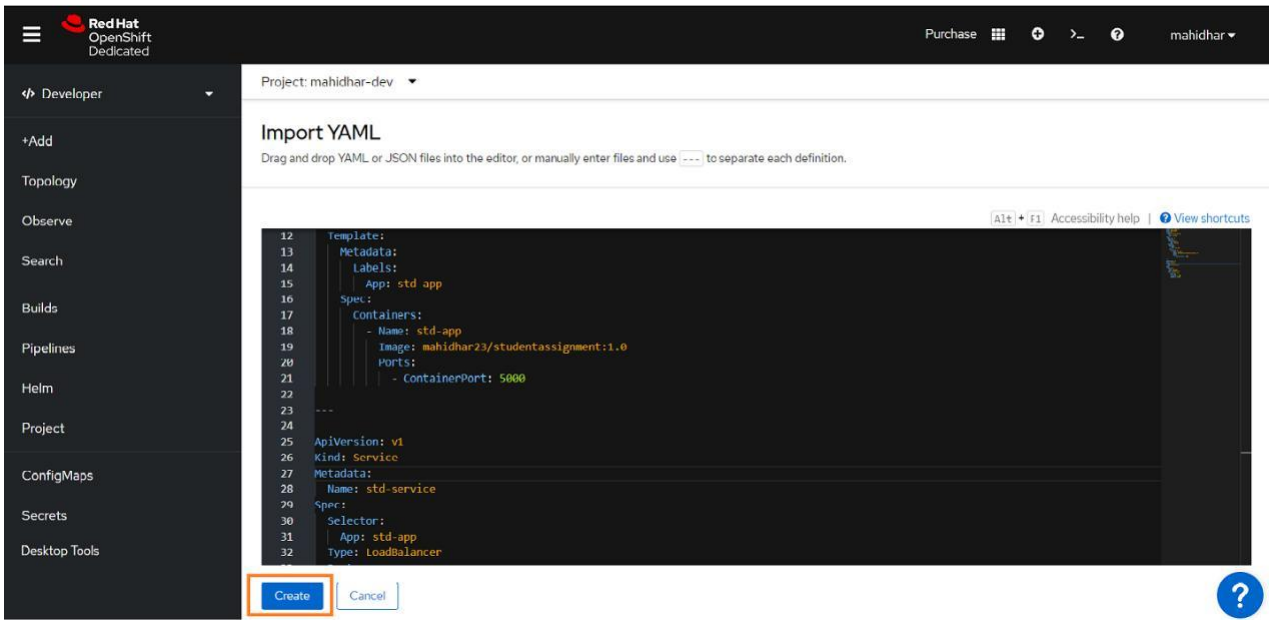
9.3

Deploying Your Application In Openshift Using YAML File

Deploying the application using the Yaml configuration file is also the same as github, go to that add button on the left side panel and select the Import YAML File From Local Machine section.

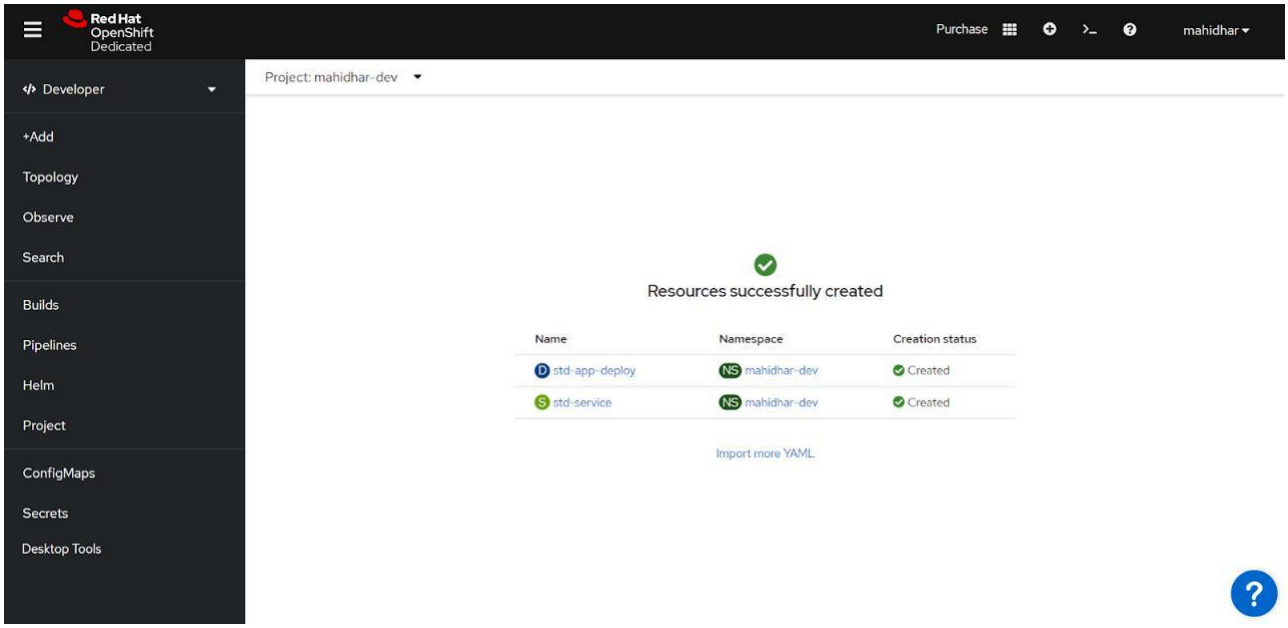


Once you click on this, you will get a blank text space, where you can paste the YAML configuration File codes of both deployment.yaml and service.yaml

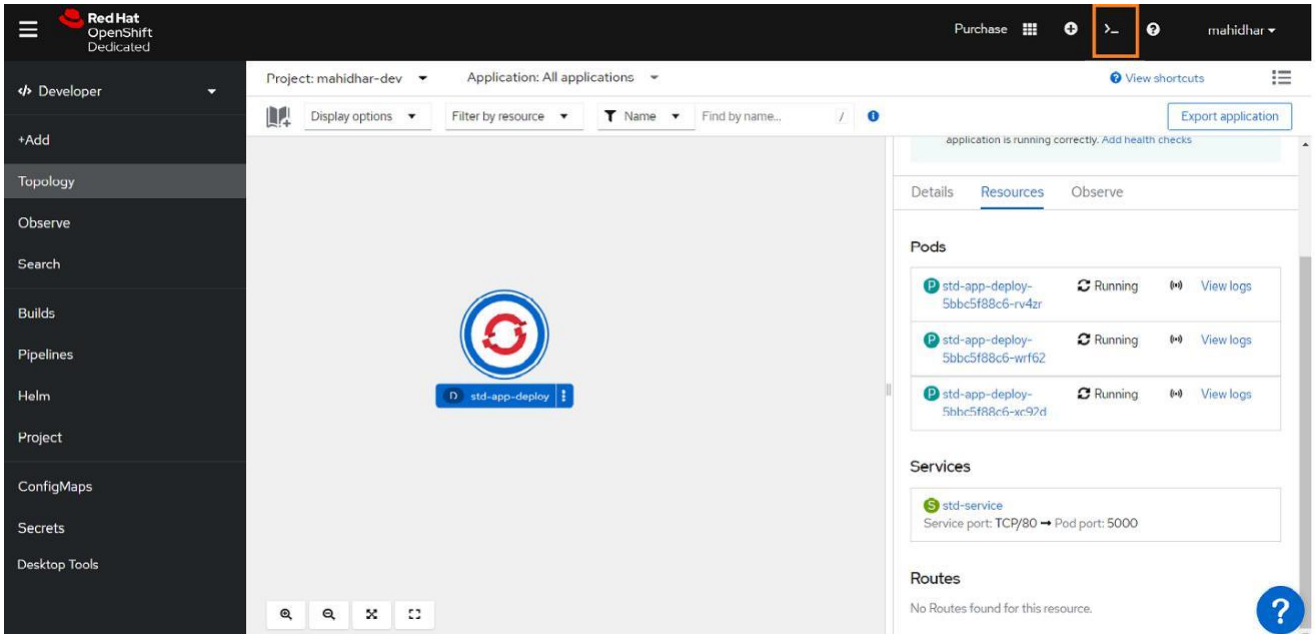


After pasting both the yaml file codes, click on create.

Note: Paste both the code in this text editor, while pasting first paste deployment then give — (three dashes, as you can see in line number 23 of the above picture) then paste the service code.

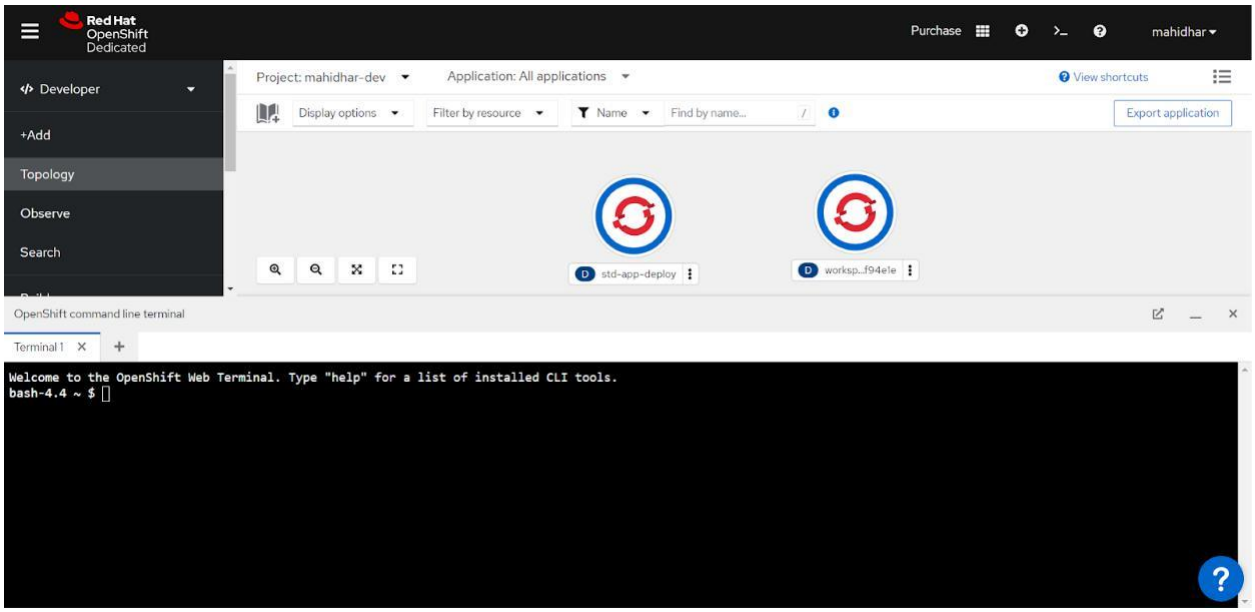


Once you click on create, if everything is good it successfully creates the deployment and service for that deployment. Now go to Topology on the left side to see the deployment.

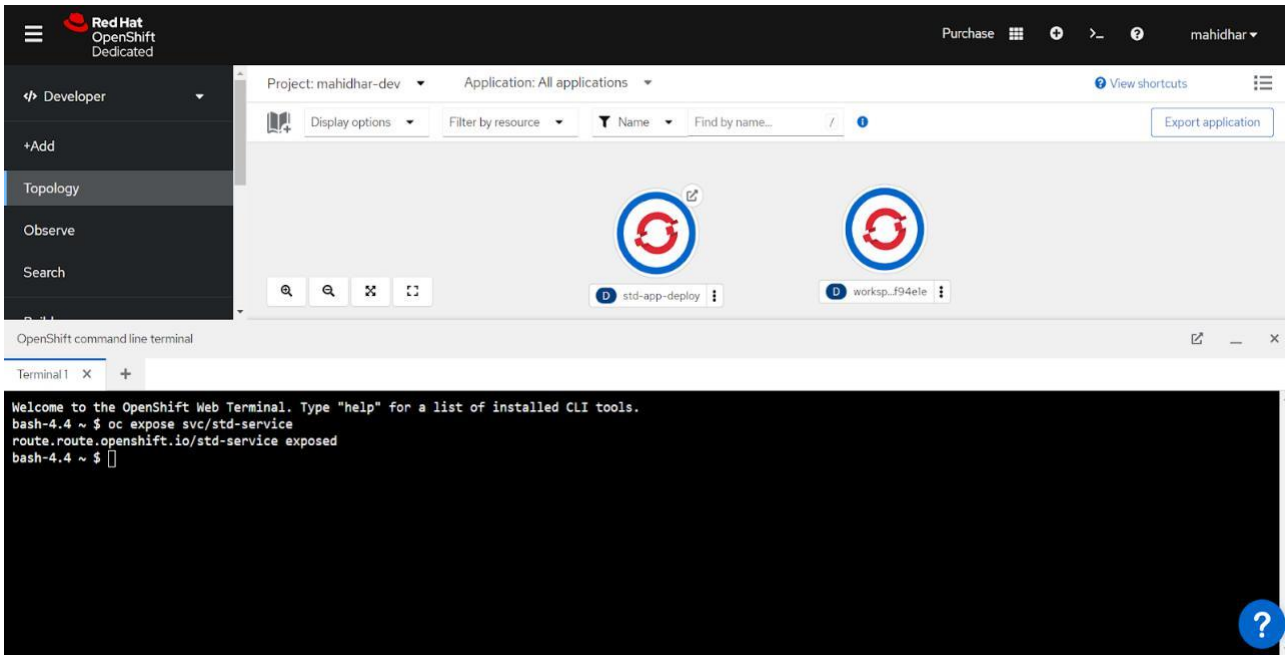


Now service and pods have been created, in service we have also created an internal service/port that pods will listen to and external port where users can interact but for external service routes has not been created (url path). To create it. Now we need to manually expose the service in the openshift terminal, to do it click on the terminal button in the navigation bar (highlighted in orange colour box

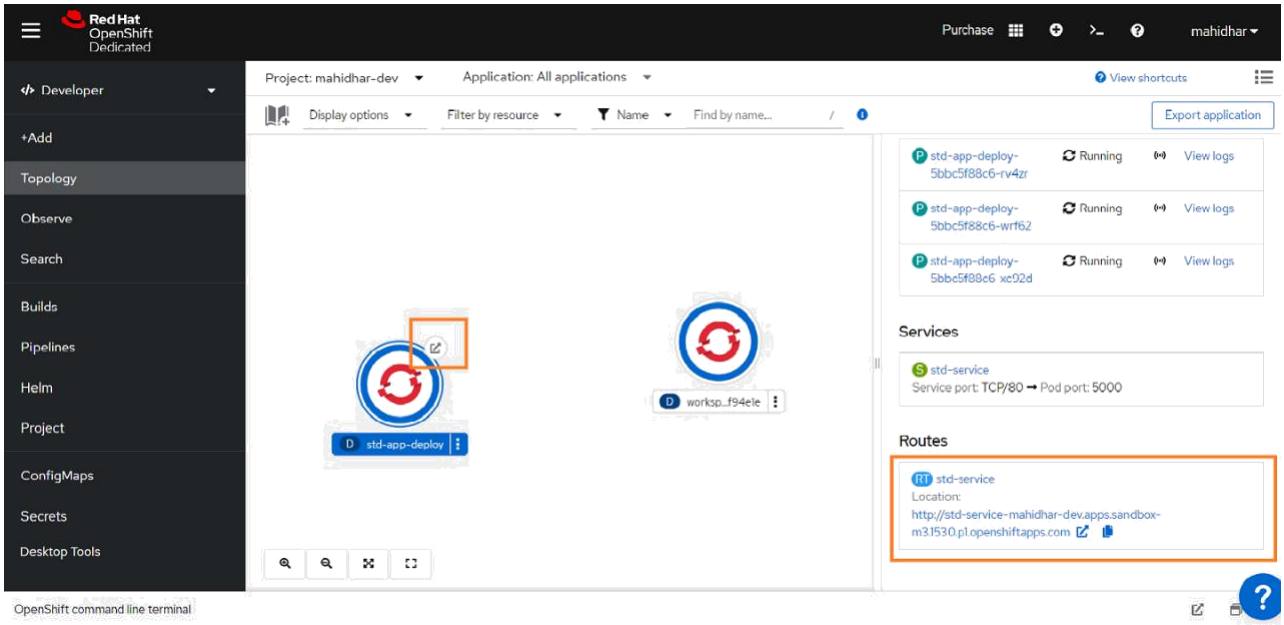
in the right top).



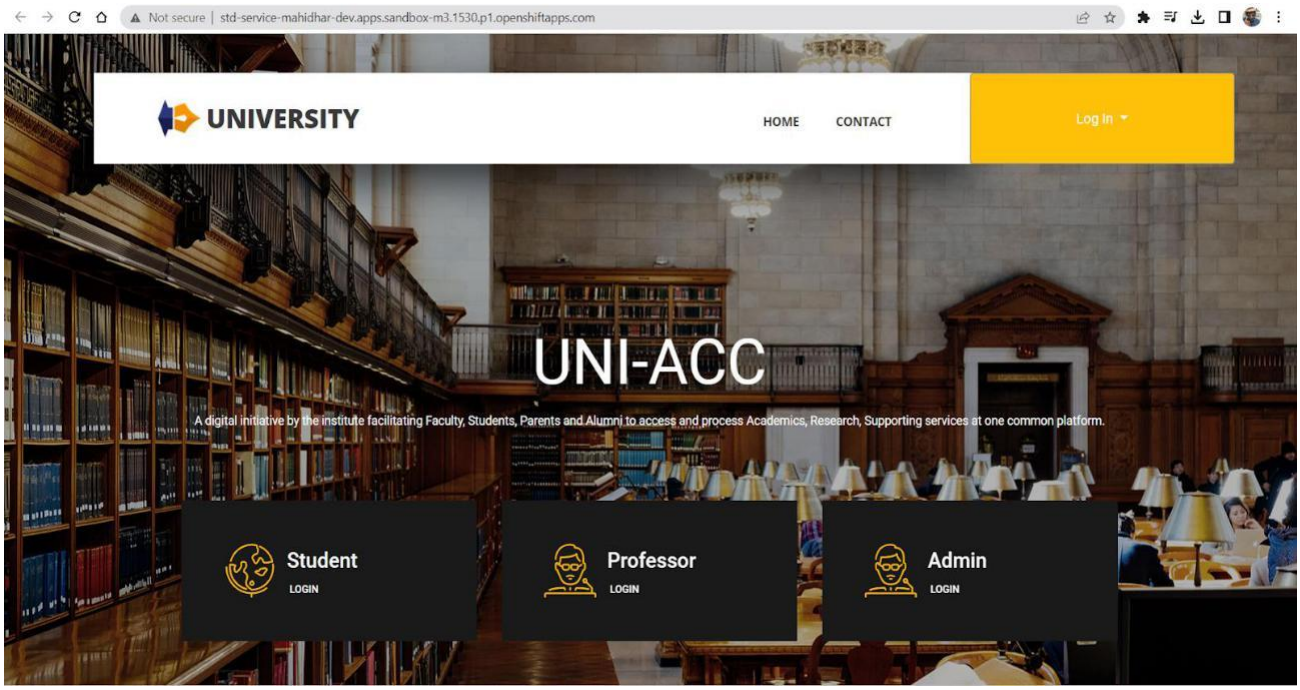
Once you click on it, it will create a terminal by creating a workspace. In this terminal we need to expose the service by its name, to do it type oc expose svc/std-service (std-service is nothing but the name of the service we have created just above)

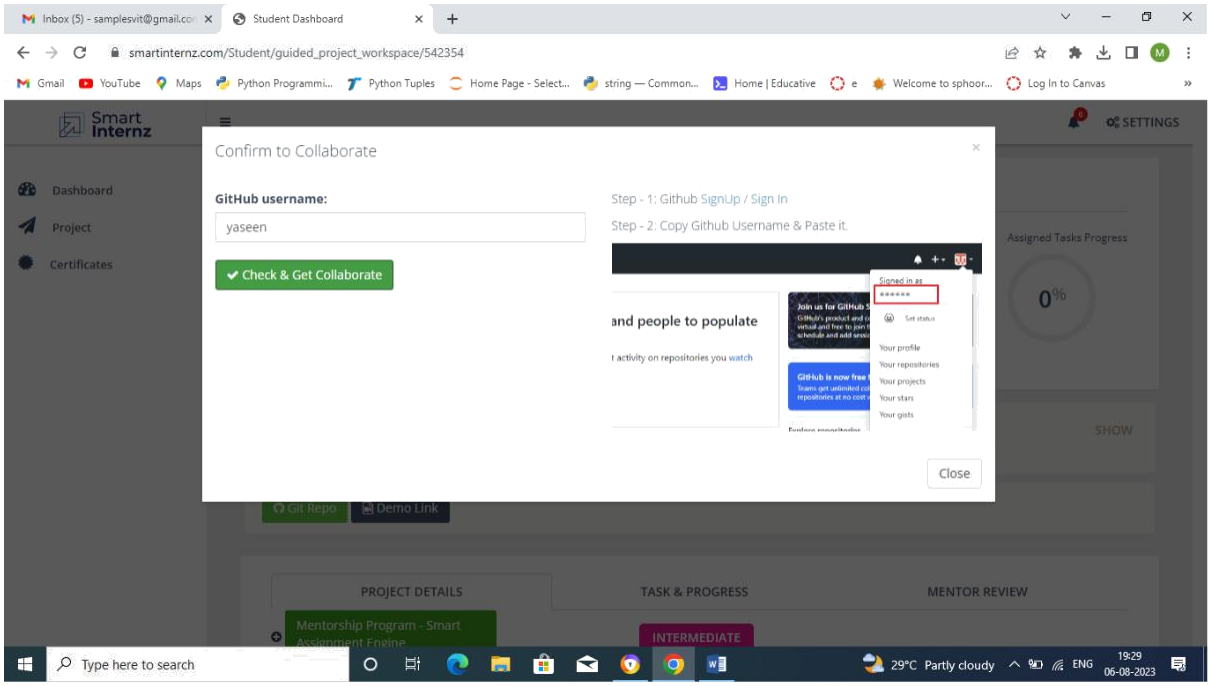


Now we have successfully exposed our service in openshift where it will create a route.



Now click on any of that, where it will take us to our application which is deployed in the openshift.




Inbox (5) - samplesvit@gmail.com
Student Dashboard
smartinternz.com/Student/guided_project_workspace/542354
Gmail
YouTube
Maps
Python Programmi...
Python Tuples
Home Page - Select...
string — Common...
Home | Educative
Welcome to sphoor...
Log In to Canvas
Smart Internz
SETTINGS
Dashboard
Project
Certificates
Confirm to Collaborate
GitHub username:
yaseen
Check & Get Collaborate
Step - 1: Github SignUp / Sign In
Step - 2: Copy Github Username & Paste it.
Close
Assigned Tasks Progress
0%
SHOW
Git Repo
Demo Link
PROJECT DETAILS
TASK & PROGRESS
MENTOR REVIEW
Mentorship Program - Smart Assignment Engine
INTERMEDIATE
Type here to search
29°C Partly cloudy
ENG
19:29
06-08-2023