

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, 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.

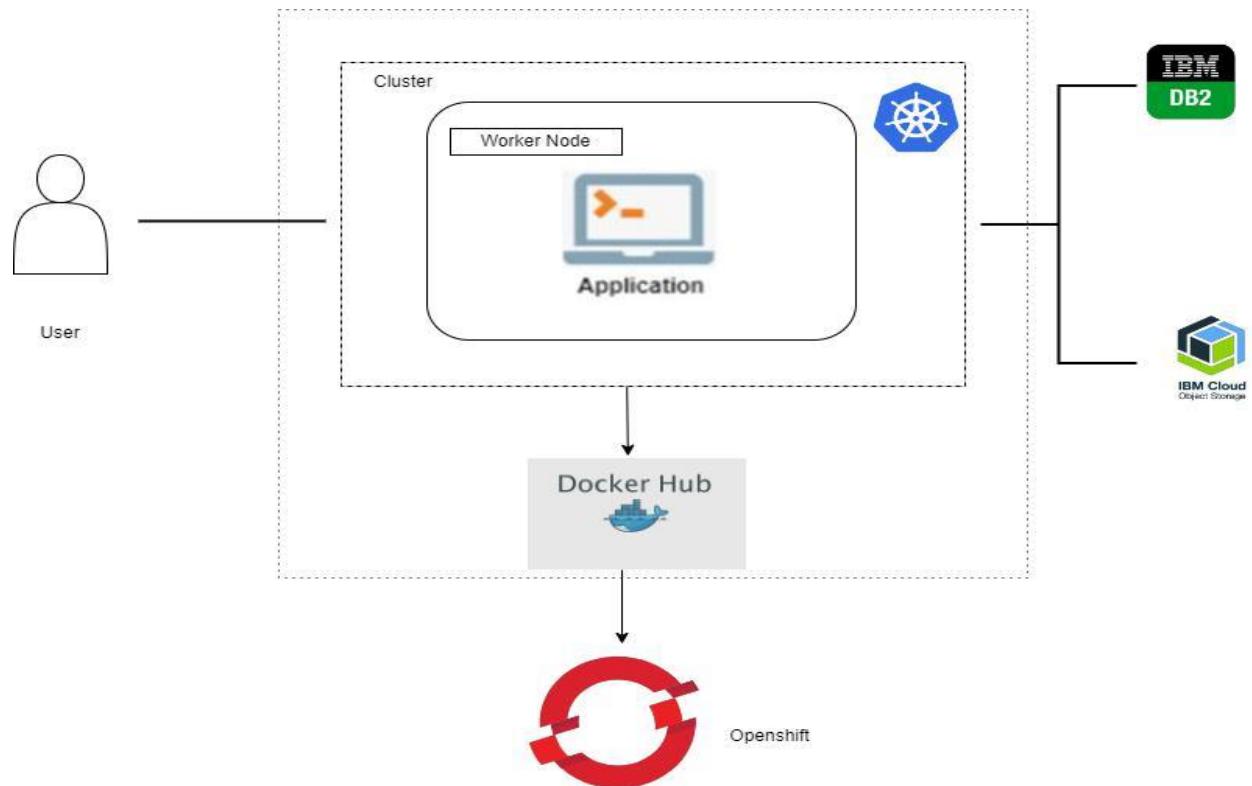
Smart Assignment aims to enhance the assignment management process, promote collaboration, and improve educational outcomes. By leveraging intelligent technologies and providing a user-friendly interface, the application streamlines administrative tasks, enhances student-teacher communication, and facilitates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

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.

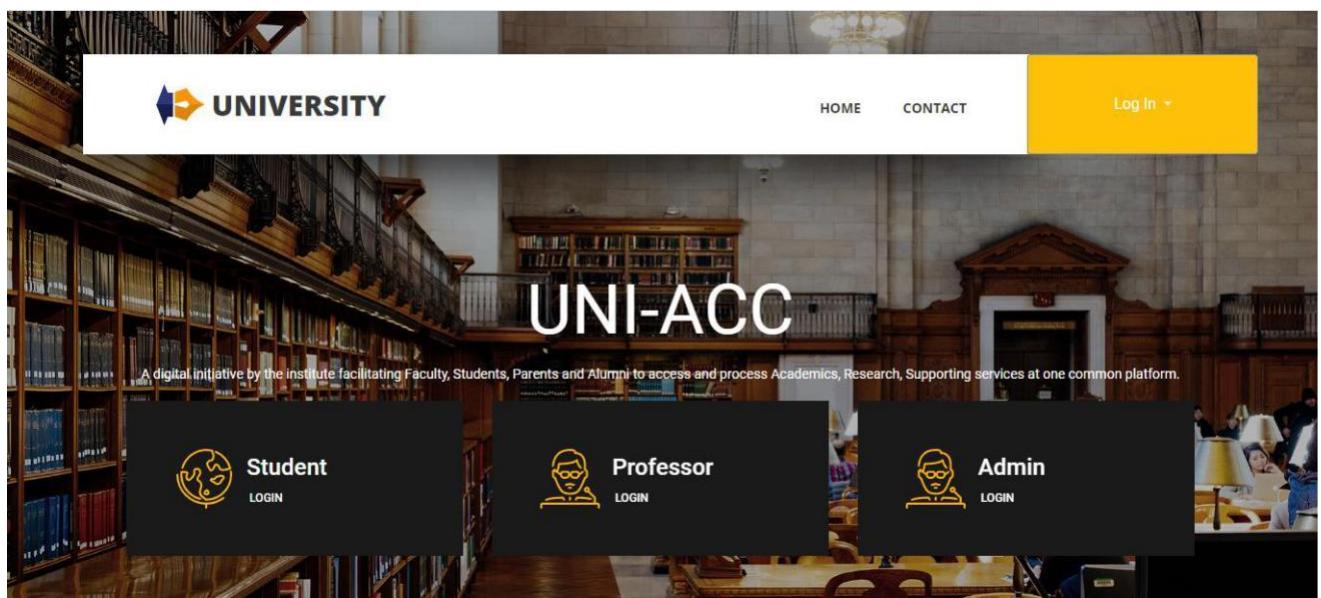
2. Creating A User Interface

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.

COURSE

HOME +91 4566778824

Name: _____

E-mail: _____

Registration Number: _____

Feedback:

SEND MESSAGE

If you are facing any issues you can fill the form that is provided or you contact CTO of the University by contact details that provided below

MR. XXXXXXXXXXXXXXXXXX
CTO

CTO Office, Administration Building

+91 3748 2445 32

cto@university.com

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.

UNIVERSITY

HOME CONTACT

Please enter your login and password!

Email: _____

Password: _____

[Forgot password?](#)

Login

Activity 1.4: adminprofile.html

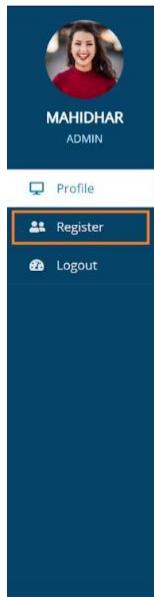
In this html page, when the admin login with their credentials with the backend code of routing we will find whether it's admin or not. If yes then we will route him from login to this html page.



name	MAHIDHAR
email	mahidhar@thesmartbridge.com
Username	mahidhar23

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



name	MAHIDHAR
email	mahidhar@thesmartbridge.com
Username	mahidhar23

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).

NAME

EMAIL

USERNAME

ROLE *

SELECT

SUBMIT

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...

name	saumya mohandas
email	saumya@thesmartbridge.com
Username	saumyan

Activity 1.7: studentsubmit.html

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

The screenshot shows a student profile page. On the left, there is a sidebar with a user icon, the name "saumya mohandas" (STUDENT), and navigation links: "Profile" (highlighted in orange), "Assignment" (selected), "Admin", and "Logout". The main content area displays a table with three rows: "name" (saumya mohandas), "email" (saumya@thesmartbridge.com), and "Username" (saumyan).

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.

The screenshot shows a list of assignments for a student. The sidebar on the left is identical to the previous one. The main content area displays a table with four rows of assignment details:

Assignment No	Assignment Name	Due Date	File	Submitted Date & Time	Marks
1	Operating System Assignment 1	25/10/2022	<input type="button" value="Choose File"/> No file chosen	<input checked="" type="checkbox"/>	
2	Operating System Assignment 2	25/11/2022	<input type="button" value="Choose File"/> No file chosen	<input checked="" type="checkbox"/>	
3	Operating System Assignment 3	25/12/2022	<input type="button" value="Choose File"/> No file chosen	<input checked="" type="checkbox"/>	
4	Operating System Assignment 4	25/01/2023	<input type="button" value="Choose File"/> No file chosen	<input checked="" type="checkbox"/>	

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

Assignment No	Assignment Name	Due Date	File	Submitted Date & Time	Marks
1	Operating System Assignment 1	25/10/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
2	Operating System Assignment 2	25/11/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
3	Operating System Assignment 3	25/12/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
4	Operating System Assignment 4	25/01/2023	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		

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.

Assignment No	Assignment Name	Due Date	File	Submitted Date & Time	Marks
1	Operating System Assignment 1	25/10/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>	2023-06-13 12:08:23	None
2	Operating System Assignment 2	25/11/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
3	Operating System Assignment 3	25/12/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
4	Operating System Assignment 4	25/01/2023	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		

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

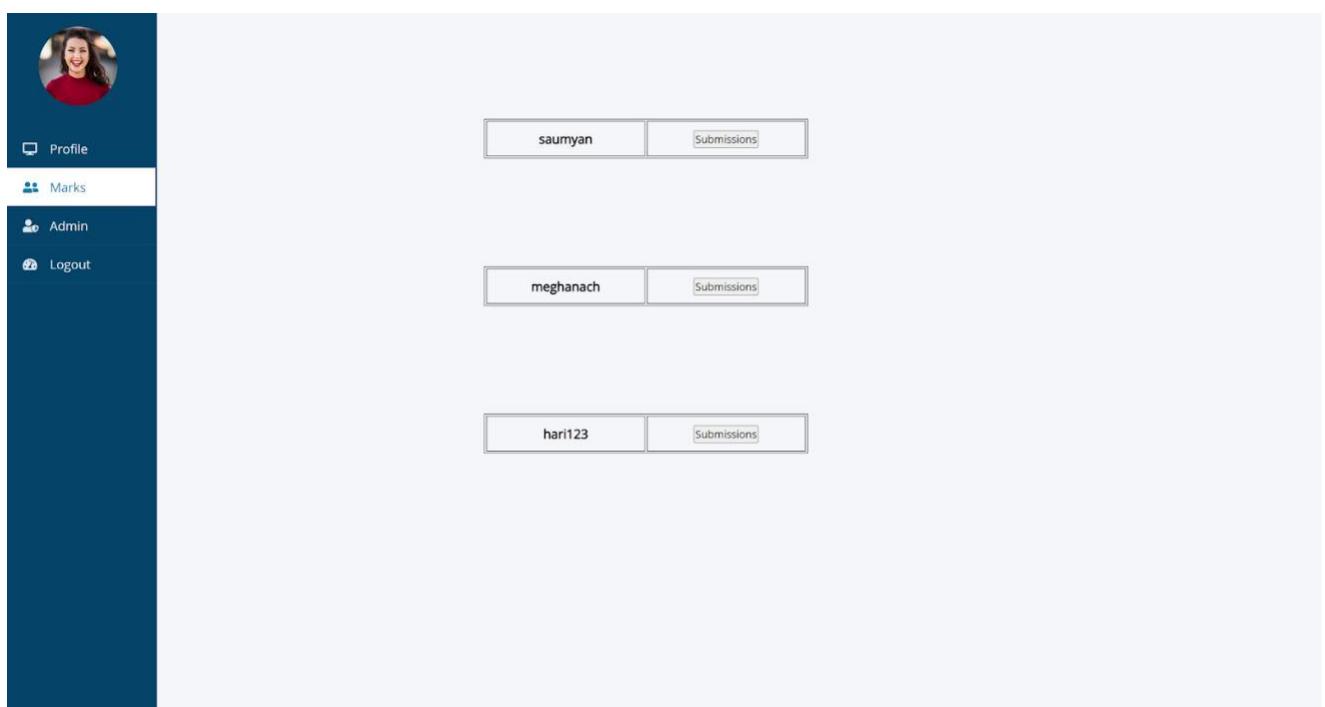
In this html page, same as other profiles pages. We will have all the information about the faculty. Here instead of making three different html pages we can make it into a single page but need to figure the info we have shown, then we can make a dynamic page regarding it. To make it more dynamic it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



name	sivakumari
email	sivakumari@gmail.com
Username	sivakumarich

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.



saumyan	Submissions
meghanach	Submissions
hari123	Submissions

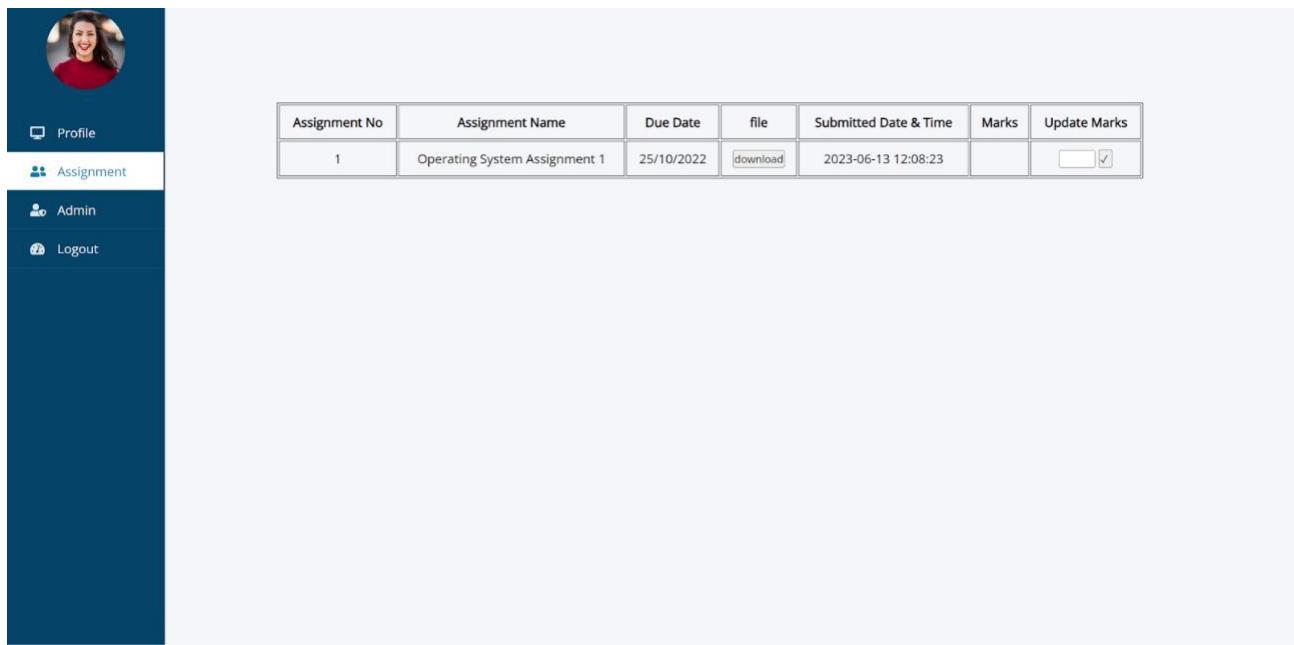
Now to see the submissions they have done, you click on the submissions button (highlighted in the orange color box in below picture)



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

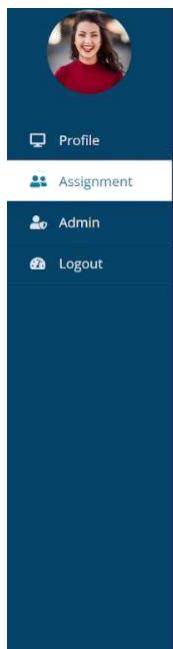
Activity 1.10: facultymarks.html

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	<input type="checkbox"/>	<input checked="" type="checkbox"/>

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

CAD Assignment 3

In the Assignment, you created 3 HTML files (index, profile and log-in), CSS files and stored the registration data in IBM db2 and did login validation.

In this assignment follow these tasks

Task 1: You have to containerize the whole application by using the docker

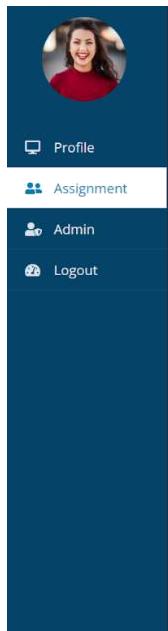
Task 2: Upload this docker image to the IBM container registry

Task 3: Then orchestrate the docker container using Kubernetes.

Submission:

1. Upload all the html, CSS, js files and Flask file then take
2. Take screenshots of the Docker desktop/Docker Playground after running the image
3. Take a screenshot of the IBM container registry after pushing the image and copy the URL and paste it below it
4. Take a screenshot of the Kubernetes dashboard and give the IP address with the port address below this so that we can publicly access it.

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



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	10	<input checked="" type="checkbox"/> 1 <input type="checkbox"/> 2

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.

MARKS UPDATED



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	10	<input type="checkbox"/> <input checked="" type="checkbox"/>

Below are the marks updated in the student account.

A screenshot of a user profile interface. At the top is a circular profile picture of a woman. Below it is a navigation bar with four items: 'Profile' (selected), 'Assignment' (highlighted in blue), 'Admin', and 'Logout'. The main content area shows a table of assignments:

Assignment No	Assignment Name	Due Date	File	Submitted Date & Time	Marks
1	Operating System Assignment 1	25/10/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>	2023-06-13 12:08:23	10
2	Operating System Assignment 2	25/11/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
3	Operating System Assignment 3	25/12/2022	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		
4	Operating System Assignment 4	25/01/2023	<input type="button" value="Choose File"/> No file chosen <input checked="" type="checkbox"/>		

3.

Database Connection

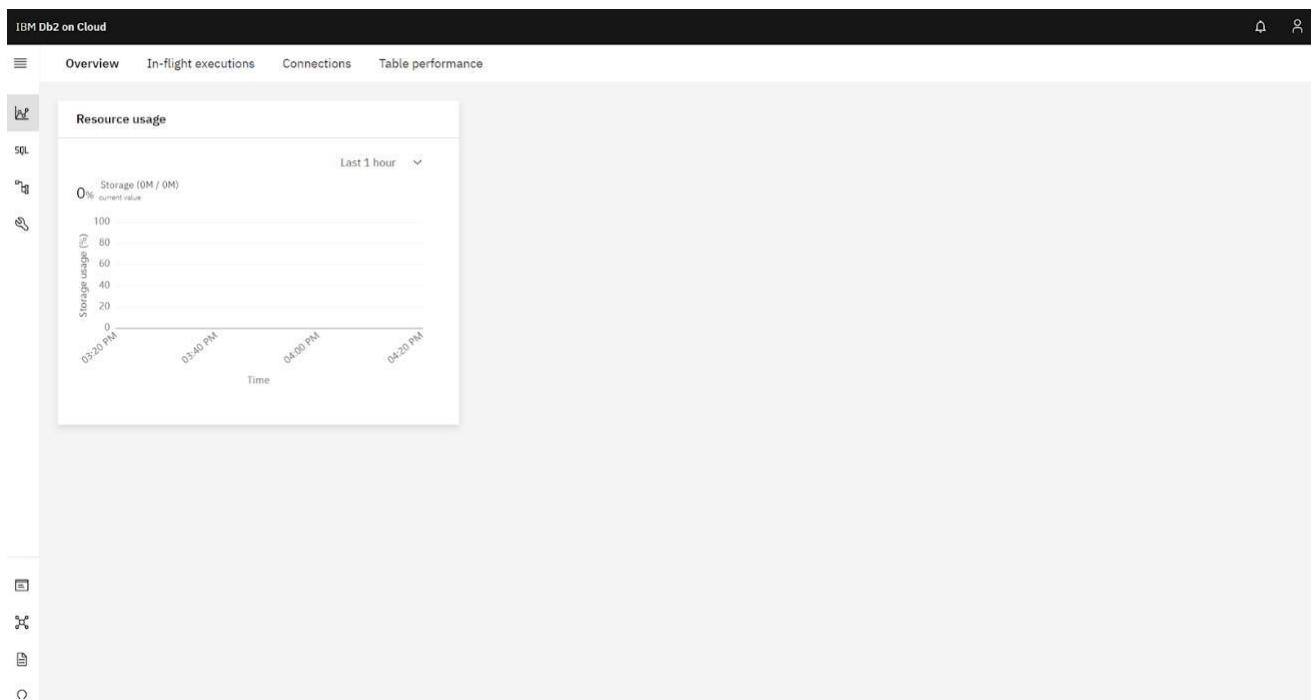
Activity 1.1: Creating Register Table

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

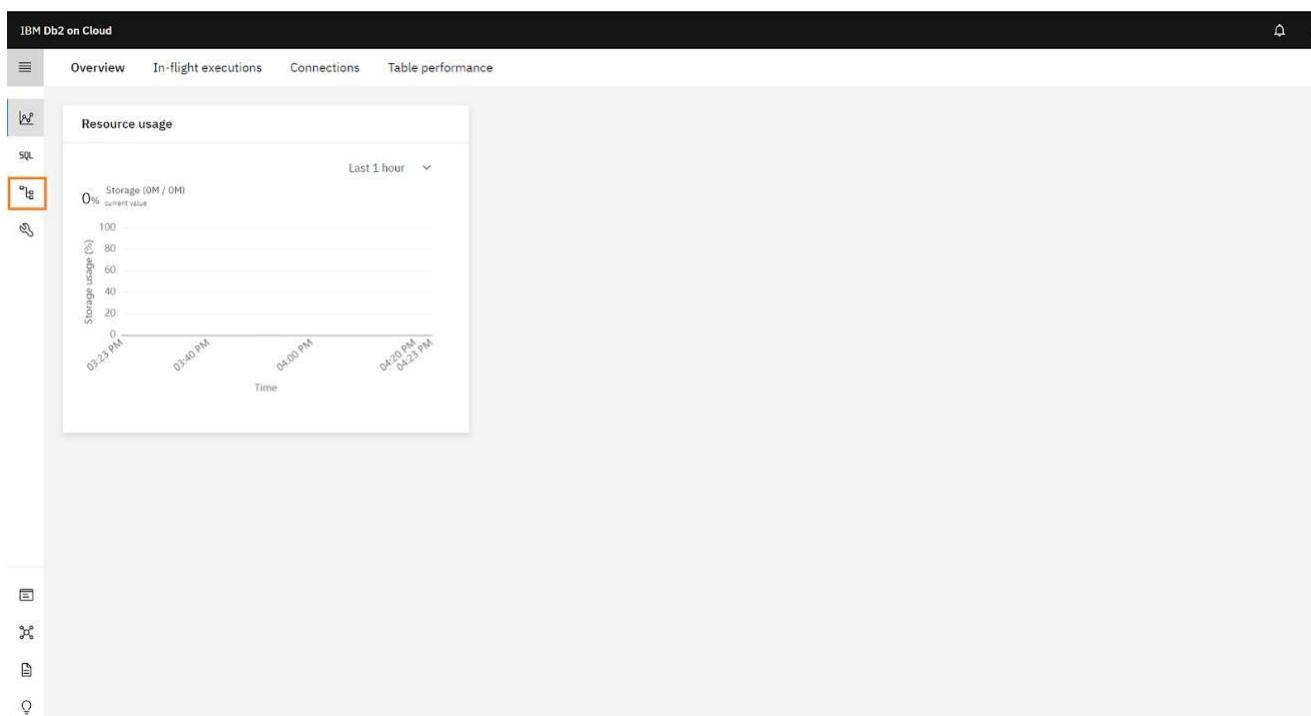
A screenshot of the IBM Cloud service details page for 'Db2-eg'. The top navigation bar shows 'Resource list / Db2-eg' with status 'Active' and 'Add tags'. On the right are 'Details' and 'Actions...' buttons. The main content area is divided into sections:

- Manage** (selected): Sub-links include 'Getting started', 'Service credentials', and 'Connections'. A red dashed box highlights the 'Go to UI' button.
- Getting started**: Includes a question about finding credentials and instructions to click 'Service Credentials' in 'Manage'.
- Need help?**: Includes a link to submit a support case.

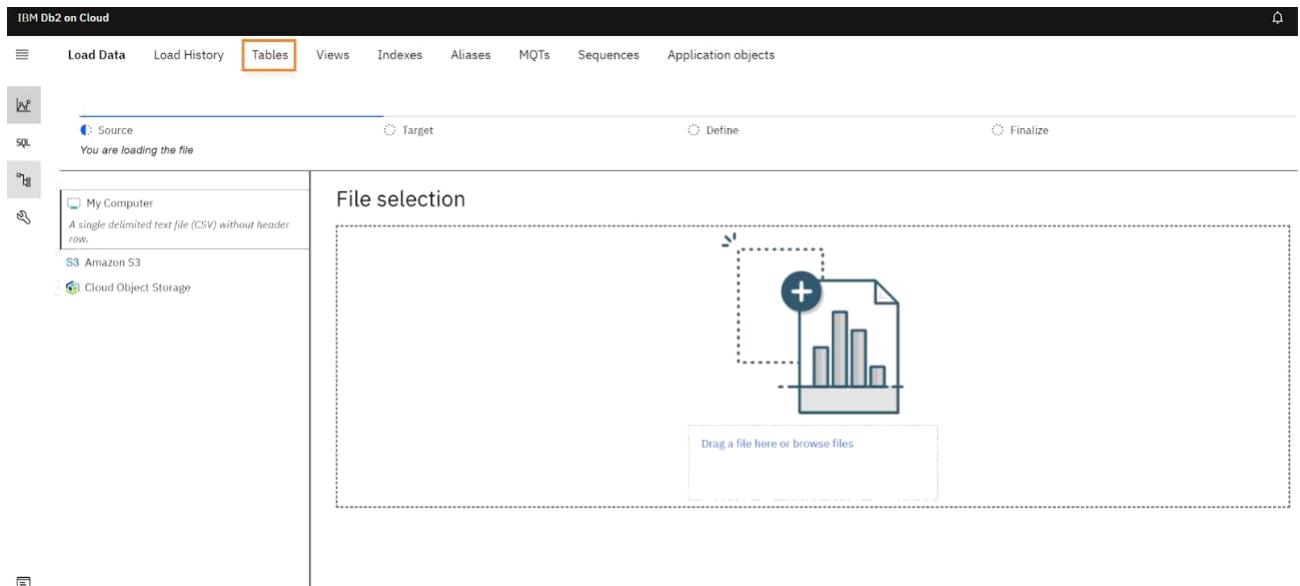
Once you click on the Go to UI button, it will take you to the db2 page.



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.

The screenshot shows the IBM Db2 on Cloud web interface. At the top, there's a navigation bar with tabs: 'Load Data', 'Load History', 'Tables' (which is highlighted with an orange border), 'Views', 'Indexes', 'Aliases', 'MQTs', 'Sequences', and 'Application objects'. Below the navigation bar, there's a search bar with placeholder text 'Find schemas or tables' and a 'Refresh' button. The main content area has a table titled 'Schemas'. The table has columns: 'Name' (checkbox), 'Definer type' (checkbox), and 'Tables' (checkbox). There is one row in the table: 'DNS98308' (checkbox), 'User' (checkbox), and '0' (checkbox). At the bottom of the main content area, there's a footer with icons for 'SQL', 'Tables', 'Views', 'Indexes', 'Aliases', 'MQTs', 'Sequences', and 'Application objects'. The footer also includes a message 'Total: 1, selected: 0'.

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

Now click on the schema name

The screenshot shows the IBM Db2 on Cloud interface. At the top, there's a navigation bar with links for Load Data, Load History, Tables, Views, Indexes, Aliases, MQTs, Sequences, and Application objects. Below this is a search bar labeled 'Find schemas or tables'. The main area is titled 'Schemas' and contains a table with columns for Name, Definer type, and Tables. One row, 'DNS98308', has a checkbox next to it, which is checked and highlighted with an orange border. The 'Tables' column shows a value of 0. On the left side, there are icons for different database objects like tables, views, and sequences. At the bottom, it says 'Total: 1, selected: 0'.

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.

This screenshot shows the same IBM Db2 on Cloud interface as before, but with a new tab open on the right labeled 'Tables'. This tab also has a 'Schemas' section with the same 'DNS98308' entry. Below the schemas, there's a table with columns for Name, Schema, and Properties. In the top right corner of this table, there's a blue button labeled 'New table' with a '+' sign, which is highlighted with an orange border. The main 'Tables' area below the schema table displays a message: 'You don't have any data currently'. At the bottom of the interface, it says 'Total: 1, selected: 1' on the left and 'Total: 0, selected: 0' on the right.

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

Table name
Enter table name

Name	Data type	Nullable	Length	Scale
COL1	CHAR	Y	5	0

Add column +

Cancel Generate DDL Create

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.

Table name
REGISTER

Name	Data type	Nullable	Length	Scale
COL1	CHAR	Y	5	0

Add column +

Cancel Generate DDL Create

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.

The screenshot shows the 'Create new table' dialog in the IBM Db2 on Cloud interface. The table name is set to 'REGS2018'. The first column, which initially had a placeholder 'COL1', now has the name 'NAME'. The column details are: Data type: CHAR, Nullable: Y, Length: 5, Scale: ..

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	5	..

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

The screenshot shows the 'Create new table' dialog in the IBM Db2 on Cloud interface. The table name is set to 'REGISTER'. The first column, which initially had a placeholder 'COL1', now has the name 'NAME'. The column details are: Data type: CHAR, Nullable: Y, Length: 5, Scale: ..

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	5	..

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

The screenshot shows the 'Create new table' dialog in the IBM Db2 on Cloud interface. The table name is set to 'REGISTER'. A single column 'NAME' is defined with a data type of CHAR and a length of 32. The 'Add column +' button is located below the table definition area. At the bottom right, there are 'Generate DDL' and 'Create' buttons.

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	..

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

The screenshot shows the 'Create new table' dialog with the 'Add column +' button highlighted by an orange box. The table name is 'REGISTER' and one column 'NAME' is defined with type CHAR(32). The 'Add column +' button is located below the table definition area. At the bottom right, there are 'Generate DDL' and 'Create' buttons.

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	..

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

IBM Db2 on Cloud

Load Data Load History Tables Views Indexes Aliases MQTs Sequences Application objects

Create new table
New table within DNS98308 schema

Table name
REGISTER

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	..
COL2	CHAR	Y	5	..

Add column +

Cancel Generate DDL Create

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

IBM Db2 on Cloud

Load Data Load History Tables Views Indexes Aliases MQTs Sequences Application objects

Create new table
New table within DNS98308 schema

Table name
REGISTER

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	..
EMAIL	VARCHAR	Y	32	..
USERNAME	VARCHAR	Y	32	..
PASSWORD	VARCHAR	Y	16	..
ROLE	INTEGER	Y

Add column +

Cancel Generate DDL Create

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

IBM Db2 on Cloud

Load Data Load History **Tables** Views Indexes Aliases MQTs Sequences Application objects

Create new table

New table within DNS98308 schema

Table name: REGISTER

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	--
EMAIL	VARCHAR	Y	32	--
USERNAME	VARCHAR	Y	32	--
PASSWORD	VARCHAR	Y	16	--
ROLE	INTEGER	Y	--	--

Add column +

Cancel Generate DDL Create

It will create a table.

IBM Db2 on Cloud

Load Data Load History **Tables** Views Indexes Aliases MQTs Sequences Application objects

Find schemas or tables Refresh

Name	Definer type	Tables
DNS98308	User	1

Name	Schema	Properties
REGISTER	DNS98308	...

Total: 1, selected: 1 Total: 1, selected: 0

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

IBM Db2 on Cloud

Load Data Load History Tables Views Indexes Aliases MQTs Sequences Application objects

Find schemas or tables Refresh

Tables

New table +

Name	Schema	Properties
REGISTER	DNS98308	...

Total: 1, selected: 0

Table definition

REGISTER

No statistics available.

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	0
EMAIL	VARCHAR	Y	32	0
USERNAME	VARCHAR	Y	32	0
PASSWORD	VARCHAR	Y	16	0
ROLE	INTEGER	Y		0

View data

On 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.

IBM Db2 on Cloud

Load Data Load History Tables Views Indexes Aliases MQTs Sequences Application objects

Find schemas or tables Refresh

Tables

New table +

Name	Schema	Properties
REGISTER	DNS98308	...

Total: 1, selected: 0

Table definition

REGISTER

No statistics available.

Name	Data type	Nullable	Length	Scale
NAME	CHAR	Y	32	0
EMAIL	VARCHAR	Y	32	0
USERNAME	VARCHAR	Y	32	0
PASSWORD	VARCHAR	Y	16	0
ROLE	INTEGER	Y		0

View data

IBM Db2 on Cloud

Load Data Load History **Tables** Views Indexes Aliases MQTs Sequences Application objects

DNS98308.REGISTER

SQL Back

NAME	EMAIL	USERNAME	PASSWORD	ROLE
There is no data here yet				

Table navigation icons: List, Database, Row, Column, Filter, Refresh, Delete, New, Edit, Copy.

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.

IBM Db2 on Cloud

Load Data Load History **Tables** Views Indexes Aliases MQTs Sequences Application objects

Create new table

New table within DNS98308 schema

Table name: SUBMIT

Name	Data type	Nullable	Length	Scale
STUDENTNAME	CHAR	Y	32	2
ASSIGNMNETNUM	INTEGER	Y	--	--
CSSUBMITTIME	TIMESTAMP	Y	10	2
MARKS	INTEGER	Y	--	--

Add column +

Cancel Generate DDL Create

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

The screenshot shows the IBM Db2 on Cloud interface. On the left, there's a sidebar with icons for Load Data, Load History, Tables, Views, Indexes, Aliases, MQTs, Sequences, and Application objects. The 'Tables' icon is highlighted. The main area has tabs for Schemas, SQL, and Views. The 'Tables' tab is selected. A search bar at the top says 'Find schemas or tables'. Below it, the 'Schemas' section shows one schema named 'DNS98308' with a user definer type and 2 tables. The 'Tables' section shows two tables: 'REGISTER' and 'SUBMIT', both in the 'DNS98308' schema. A 'Newtable' button is visible in the top right of the tables section. At the bottom, status messages say 'Total: 1, selected: 1' for the schemas and 'Total: 2, selected: 0' for the tables.

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

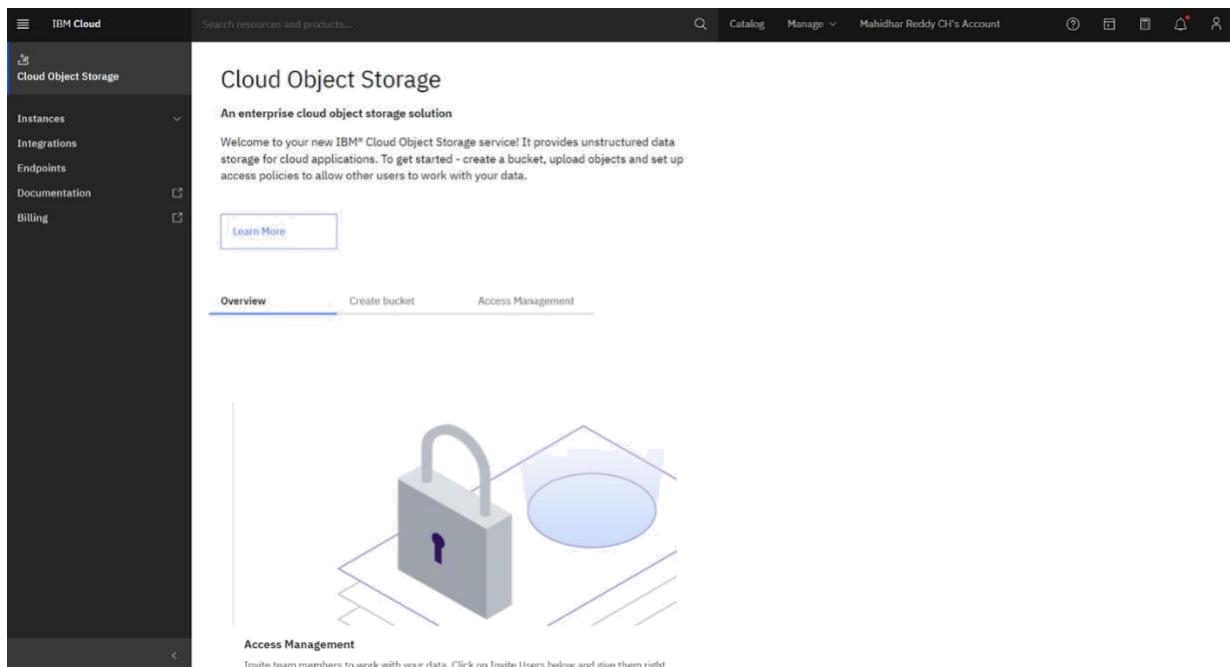
When you create an Object storage service, you will be redirected to the resources list. In that click on storage.

The screenshot shows the IBM Cloud Resource list interface. On the left, there's a sidebar with various service icons like Compute, Containers, Networking, Storage, etc. The 'Storage' section is expanded, showing sub-options like Converged infrastructure, Enterprise applications, AI / Machine Learning, Analytics, Blockchain, Databases, Developer tools, Logging and monitoring, Migration, Integration, Internet of Things, Security, and Mobile. A single item under 'Storage' is highlighted with a red box. The main area is a table with columns: Name, Group, Location, Product, Status, and Tags. There are search and filter bars at the top of each column.

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

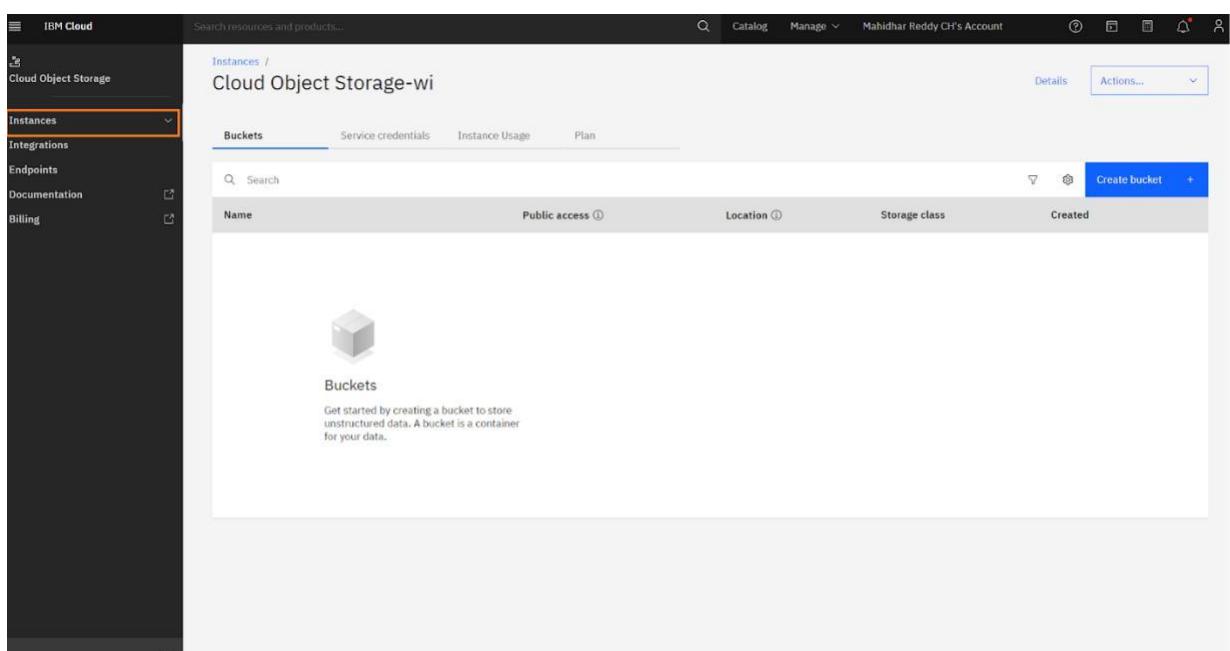
This screenshot shows the same IBM Cloud Resource list interface, but now the 'Cloud Object Storage-wi' resource under the Storage category is selected and expanded. The table row for this resource includes columns for Name (Cloud Object Storage-wi), Group (Default), Location (Global), Product (Cloud Object Storage), Status (Active), and Tags (cpdaas). The 'Cloud Object Storage-wi' entry is also highlighted with a red box.

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



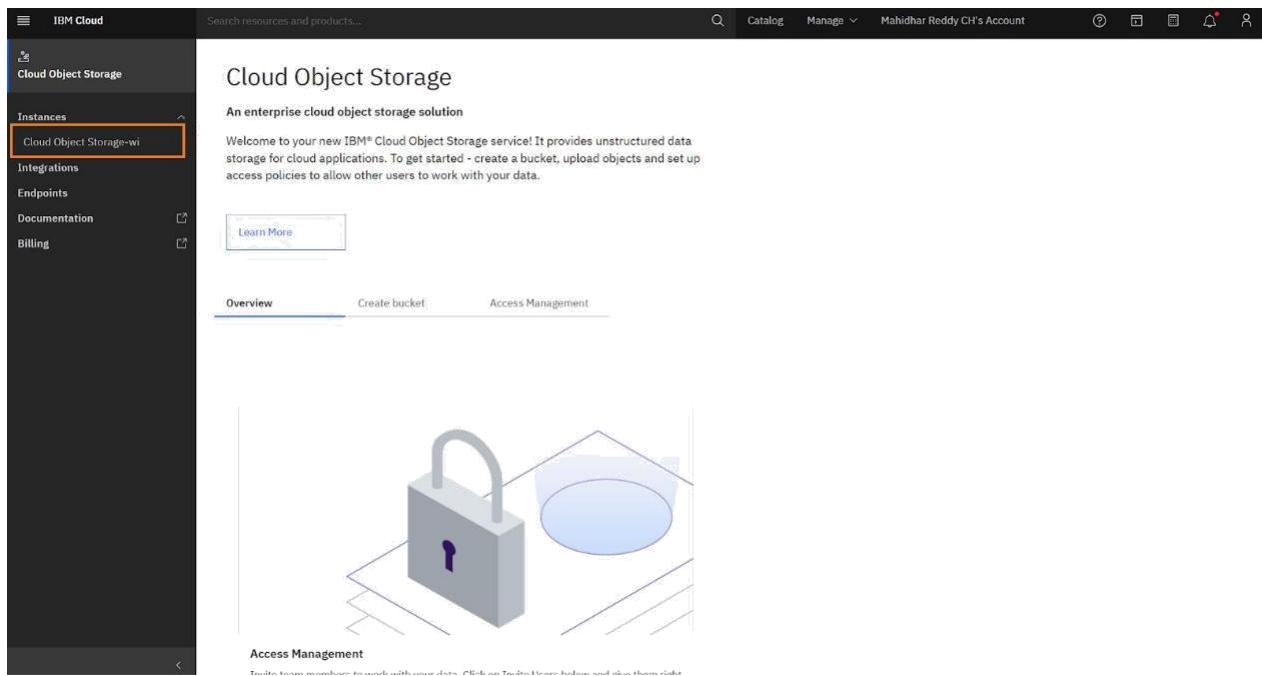
The screenshot shows the IBM Cloud Cloud Object Storage landing page. The left sidebar has a dark theme with the 'Cloud Object Storage' section selected. The main content area has a light background. At the top, there's a search bar and navigation links for Catalog, Manage, and account info. Below the search bar, the title 'Cloud Object Storage' is displayed, followed by a sub-headline 'An enterprise cloud object storage solution'. A welcome message and a 'Learn More' button are present. Below the message, there are three tabs: 'Overview' (which is active), 'Create bucket', and 'Access Management'. The central part of the page features a graphic of a padlock and a house, symbolizing security and access management. Under the 'Access Management' heading, there's a note about inviting team members.

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



The screenshot shows the 'Instances' page for the Cloud Object Storage service. The left sidebar is identical to the previous page. The main content area shows the 'Cloud Object Storage-wi' instance details. The 'Buckets' tab is selected, showing a single bucket icon. There are tabs for 'Service credentials', 'Instance Usage', and 'Plan'. A search bar and a 'Create bucket' button are at the top of the list table. The table columns are 'Name', 'Public access', 'Location', 'Storage class', and 'Created'. A note below the table explains what a bucket is.

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

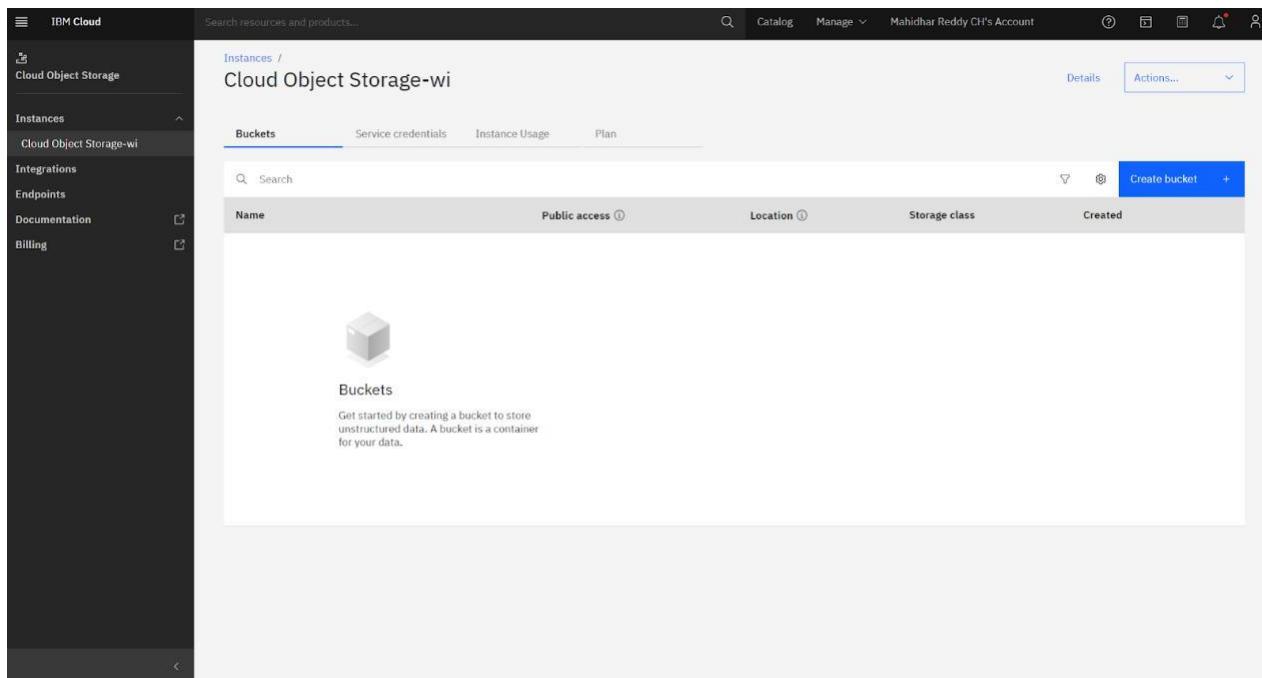


The screenshot shows the IBM Cloud interface for Cloud Object Storage. The left sidebar has a dark theme with the following navigation items:

- Cloud Object Storage
- Instances (highlighted with a red box)
- Integrations
- Endpoints
- Documentation
- Billing

The main content area is titled "Cloud Object Storage" and describes it as an enterprise cloud object storage solution. It includes a welcome message, a "Learn More" button, and tabs for "Overview", "Create bucket", and "Access Management". A large graphic of a padlock on a wireframe surface is displayed, with a blue oval shape nearby. Below the graphic, there is a section titled "Access Management" with a note about granting access to buckets.

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



The screenshot shows the "Instances" page for the "Cloud Object Storage-wi" instance. The left sidebar is identical to the previous screenshot. The main content area shows the "Buckets" tab selected, displaying a table with columns: Name, Public access, Location, Storage class, and Created. A single bucket entry is shown with a small icon and the name "Bucket". At the top right of the table, there is a "Create bucket" button. The top right corner of the interface shows account information: "Mahidhar Reddy CH's Account" and various status icons.

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

The screenshot shows the IBM Cloud Cloud Object Storage Instances page. On the left, there's a sidebar with options like Instances, Integrations, Endpoints, Documentation, and Billing. The main area is titled 'Cloud Object Storage-wi' and shows a table of 'Buckets'. At the top right of the table, there's a blue button labeled 'Create bucket' with a '+' sign. This button is highlighted with a red box.

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

The screenshot shows the 'Create bucket' page. The left sidebar has 'Cloud Object Storage-wi' selected. The main content area has a heading 'Create bucket'. Below it, there's a section titled 'Custom bucket' with a sub-section 'Customize your bucket'. This 'Customize your bucket' section is highlighted with a red box and contains the text: 'Create a bucket by selecting bucket configurations that meet your object storage needs.' There are two numbered arrows: '1' pointing to the 'Custom bucket' section and '2' pointing to the 'Customize your bucket' sub-section. Below this, there are three sections: 'Predefined buckets', 'Quickly get started', 'Archive your data', and 'Host a static website', each with a 'Show more' link and a right-pointing arrow.

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

The screenshot shows the 'Create custom bucket' page in the IBM Cloud Object Storage interface. The 'Unique bucket name' field is highlighted with an orange border. A tooltip window titled 'Bucket naming rules:' provides the following guidelines:

- Must be unique across the **whole** IBM Cloud Object Storage system
- Do not use any personal information (any part of a name, address, financial or security accounts or SSN)
- Must start and end in alphanumeric characters (3 to 63)
- Characters allowed: lowercase, numbers and nonconsecutive dots and hyphens

Below the naming rules, there are sections for 'Resiliency' (set to 'Regional'), 'Location' (set to 'jp-tok'), and 'Storage class' (set to 'Smart Tier'). The 'Create bucket' button is visible at the bottom right.

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

The screenshot shows the 'Create custom bucket' page with the 'Unique bucket name' field populated with 'studentassignment'. The 'Create bucket' button is highlighted with an orange border. The rest of the interface is identical to the first screenshot, including the 'Bucket naming rules' tooltip and the 'Smart Tier' storage class selection.

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

The screenshot shows the IBM Cloud interface for Cloud Object Storage. The left sidebar has 'Cloud Object Storage' selected under 'Instances'. The main area shows a table of buckets. One row is highlighted with a blue background, showing a bucket named 'studentassignment'. The table columns are: Name, Public access, Location, Storage class, and Created. The 'studentassignment' row has 'No' in Public access, 'jp-tok' in Location, 'Smart Tier' in Storage class, and '2023-06-14 3:07 PM' in Created.

Name	Public access	Location	Storage class	Created
studentassignment	No	jp-tok	Smart Tier	2023-06-14 3:07 PM

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

The screenshot shows the same IBM Cloud interface as before, but with a difference: the bucket name 'studentassignment' in the first row of the table is now highlighted with a red rectangular box. All other elements of the interface remain the same.

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

The screenshot shows the IBM Cloud Cloud Object Storage interface. On the left, a sidebar lists 'Instances' (Cloud Object Storage-wi), 'Integrations', 'Endpoints', and 'Billing'. The main area shows a 'studentassignment' instance under 'Cloud Object Storage-wi'. The 'Objects' tab is selected, indicated by a blue underline. A search bar at the top says 'Search resources and products...'. Below it, a breadcrumb navigation shows 'Instances / Cloud Object Storage-wi / studentassignment'. On the right, there are tabs for 'Transfers', 'Details', and 'Actions...'. The main content area displays a table with columns for 'Object name', 'Archived', 'Size', and 'Last modified'. A large 'Upload' button is visible in the top right corner of the table header. Below the table, there's a section for dragging files to upload, with a placeholder text 'Drag and drop files (objects) here or click to upload'.

Now go to the permissions tab by clicking on it.

This screenshot is identical to the previous one, but the 'Permissions' tab is now selected, indicated by an orange border around the tab name. The rest of the interface, including the sidebar, search bar, breadcrumb navigation, and main content area, remains the same.

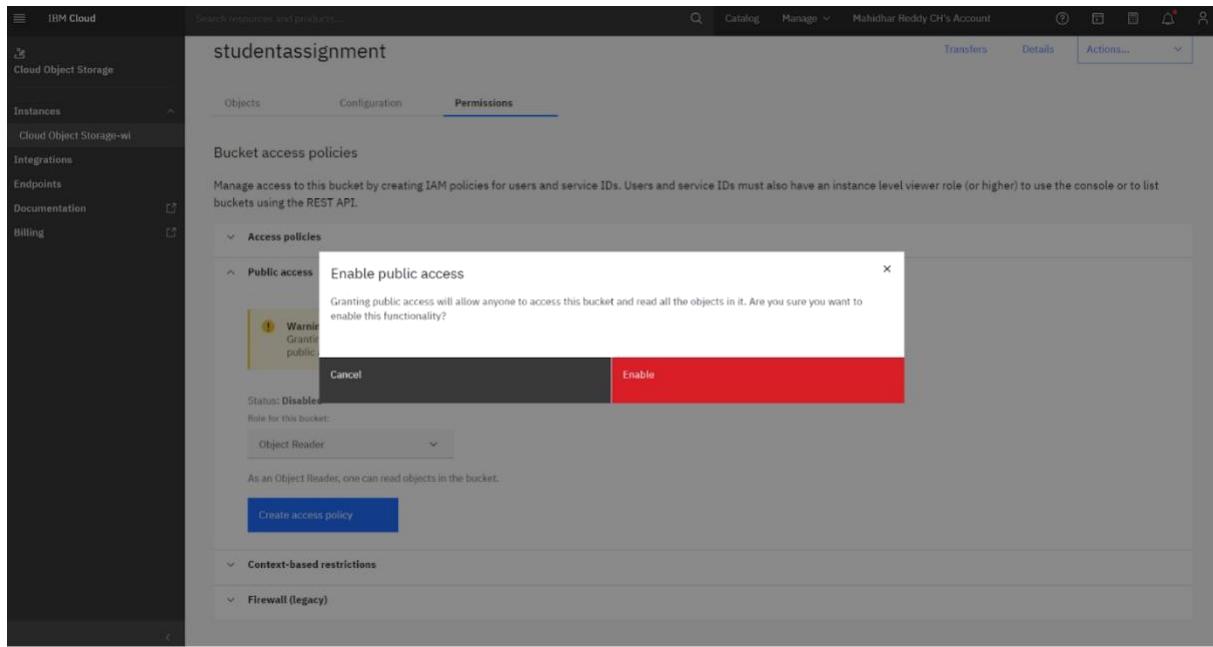
Now click on the public access drop down to change the permission settings.

The screenshot shows the IBM Cloud interface for managing Cloud Object Storage. The left sidebar is collapsed, showing 'Cloud Object Storage' under 'Instances'. The main area displays the 'studentassignment' bucket. The 'Permissions' tab is selected. Under 'Bucket access policies', the 'Public access' section is expanded and highlighted with a red box. Other sections like 'Access policies', 'Context-based restrictions', and 'Firewall (legacy)' are shown below.

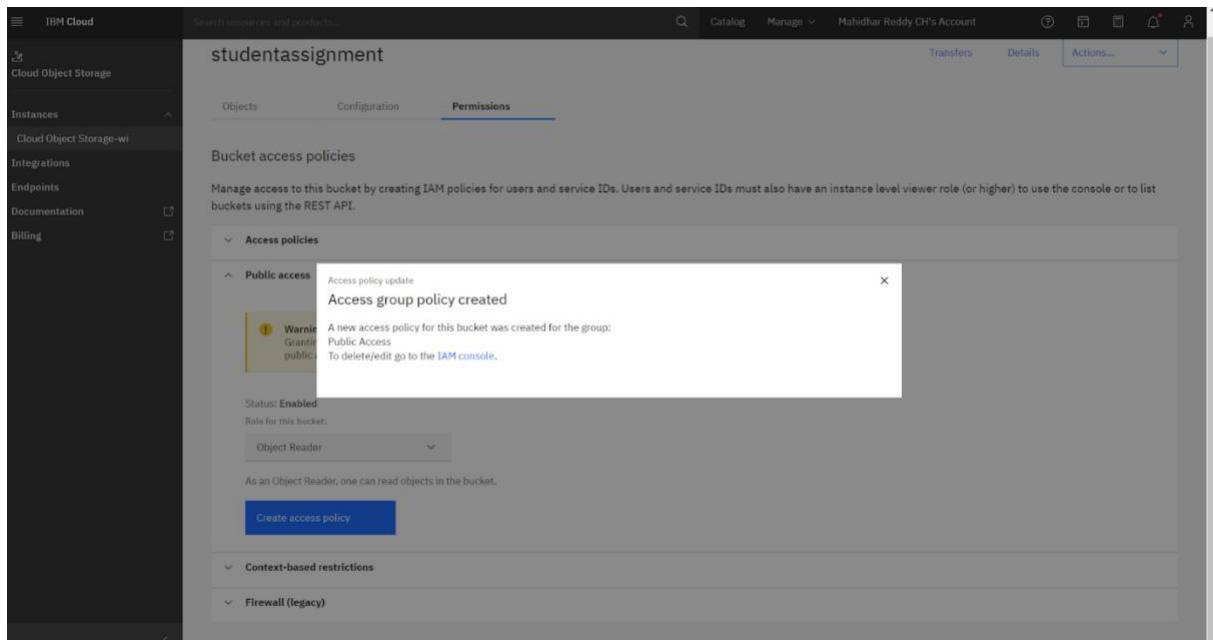
Now click on the Content Reader dropdown and select object reader (1). Now click on create access policy (2).

The screenshot shows the same IBM Cloud interface for the 'studentassignment' bucket. A yellow warning box is present, stating: 'Warning: Granting Public access to this bucket will allow anyone to access the bucket. To revoke public access, remove the "Public access" policy from this bucket within Access groups'. Below this, the 'Role for this bucket' dropdown is set to 'Object Reader' (highlighted with a red box) and step 1 is indicated. Step 2 is indicated by a red box around the 'Create access policy' button. The other sections like 'Status: Disabled', 'As an Object Reader, one can read objects in the bucket.', 'Context-based restrictions', and 'Firewall (legacy)' are also visible.

Now click on enable to change the permissions.



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



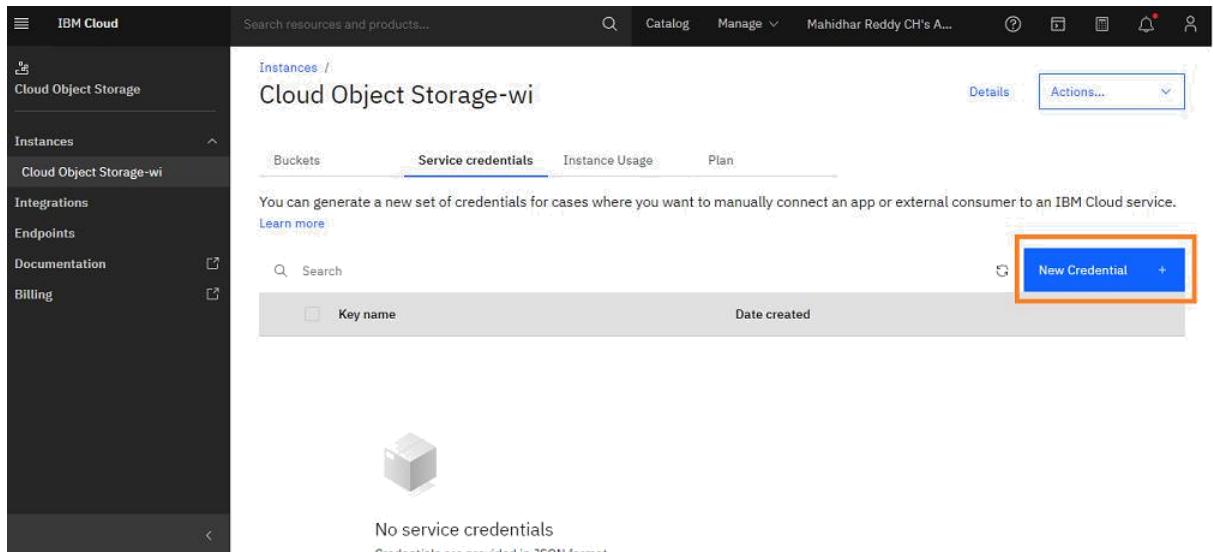
Now click on cloud object storage on the left side panel, in the dropdown of instances. So you will be redirected to the below page.

The screenshot shows the IBM Cloud interface for Cloud Object Storage. The left sidebar is dark with white text, showing 'Cloud Object Storage' under 'Instances'. The main area has a light background. At the top, there's a search bar and navigation links for Catalog, Manage, and user info. Below the search bar, it says 'Instances / Cloud Object Storage-wi'. There are four tabs: 'Buckets' (which is active and highlighted in blue), 'Service credentials', 'Instance Usage', and 'Plan'. A 'Create bucket' button is at the top right. The 'Buckets' table has columns: Name, Public access, Location, Storage class, and Created. One row is shown: 'studentassignment', Yes, jp-tok, Smart Tier, 2023-06-14 3:07 PM.

Now again click on the service credentials tab.

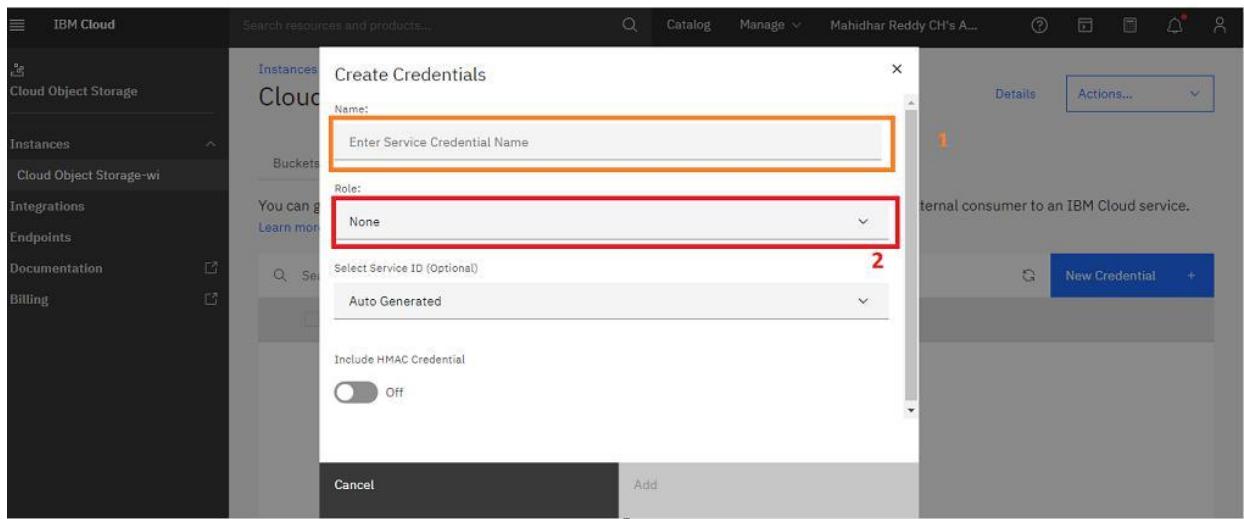
This screenshot is similar to the previous one, but the 'Service credentials' tab is now highlighted with a red box. The rest of the interface is identical, showing the same bucket information and the 'Create bucket' button.

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



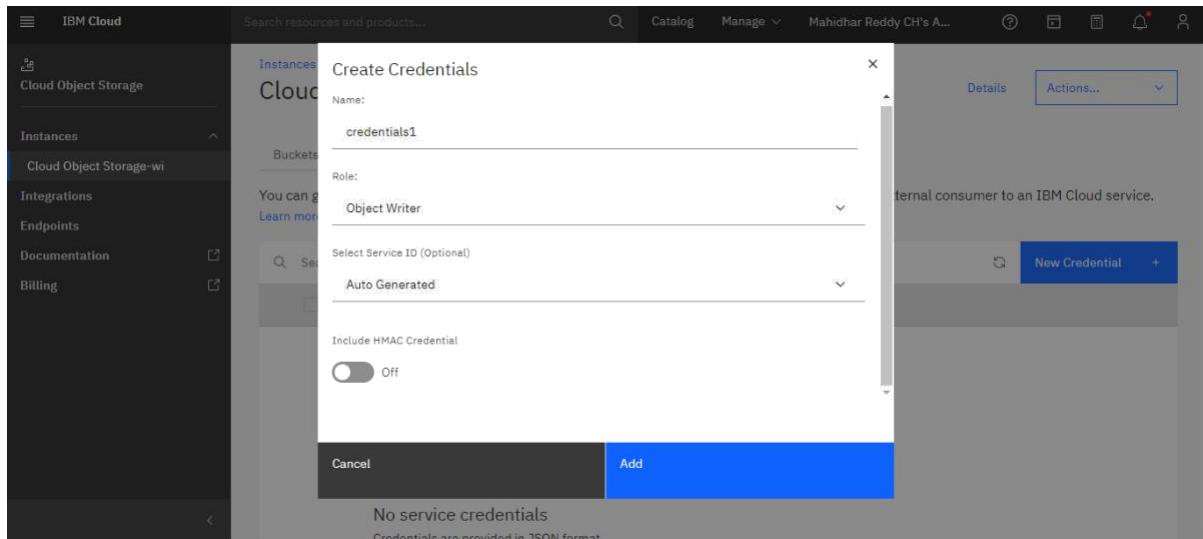
The screenshot shows the IBM Cloud interface for managing Cloud Object Storage. On the left, there's a sidebar with options like Instances, Integrations, Endpoints, Documentation, and Billing. The main area is titled 'Cloud Object Storage-wi' under 'Instances'. It has tabs for Buckets, Service credentials (which is selected), Instance Usage, and Plan. A message says, 'You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service.' Below this is a search bar and a table with columns 'Key name' and 'Date created'. At the bottom, it says 'No service credentials'. A blue button labeled 'New Credential' is highlighted with a red box.

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



The screenshot shows a 'Create Credentials' dialog box. It has fields for 'Name' (with placeholder 'Enter Service Credential Name') and 'Role' (with placeholder 'None'). There are also optional fields for 'Select Service ID (Optional)' (set to 'Auto Generated') and 'Include HMAC Credential' (set to 'Off'). Buttons at the bottom include 'Cancel' and 'Add'. The 'Name' field is highlighted with a red box (labeled 1) and the 'Role' dropdown is highlighted with a red box (labeled 2).

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

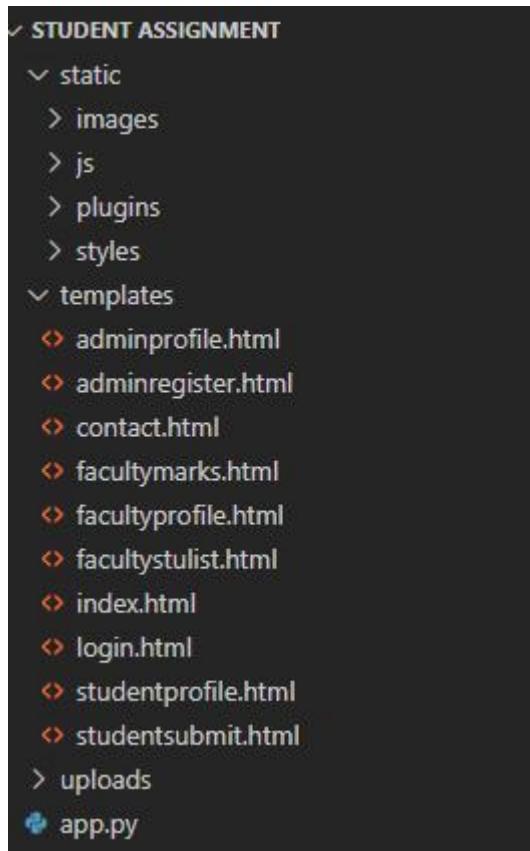


Give the name as credential 1 (we can give any name) and the role as Object Writer (Because we will read and write the objects both. Now click on Add in the bottom.

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.

```
1  from flask import Flask, render_template, request, session
2  import ibm_db
3  import ibm_boto3
4  from ibm_botocore.client import Config, ClientError
5  import os
6  import re
7  import random
8  import string
9  import datetime
10 import requests
11
12
13 app = Flask(__name__)
14 [REDACTED]
15
16 if __name__ == "__main__":
17     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.

```
17 @app.route("/")
18 def index():
19     return render_template("index.html")
```

3.5

Rendering Contact.Html Page

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

3.6

Rendering Login.Html Page

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

```

40 @app.route("/login", methods=['POST','GET'])
41 def loginentered():
42     global Userid
43     global Username
44     msg = ''
45     if request.method == "POST":
46         email = str(request.form['email'])
47         print(email)
48         password = request.form["password"]
49         sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND PASSWORD=?" # from db2 sql table
50         stmt = ibm_db.prepare(conn, sql)
51         # this username & password is should be same as db-2 details & order also
52         ibm_db.bind_param(stmt, 1, email)
53         ibm_db.bind_param(stmt, 2, password)
54         ibm_db.execute(stmt)
55         account = ibm_db.fetch_assoc(stmt)
56         print(account)
57         if account:
58             session['Loggedin'] = True
59             session['id'] = account['EMAIL']
60             Userid = account['EMAIL']
61             session['email'] = account['EMAIL']
62             Username = account['USERNAME']
63             Name = account['NAME']
64             msg = "logged in successfully !"
65             sql = "SELECT ROLE FROM register where email = ?"
66             stmt = ibm_db.prepare(conn, sql)
67             ibm_db.bind_param(stmt, 1, email)
68             ibm_db.execute(stmt)
69             r = ibm_db.fetch_assoc(stmt)
70             print(r)
71             if r['ROLE'] == 1:
72                 print("STUDENT")
73                 return render_template("studentprofile.html", msg=msg, user=email, name = Name, role= "STUDENT", username=Username, email = email)
74             elif r['ROLE'] == 2:
75                 print("FACULTY")
76                 return render_template("facultyprofile.html", msg=msg, user=email, name = Name, role= "FACULTY", username=Username, email = email)
77             else:
78                 return render_template('adminprofile.html', msg=msg, user=email, name = Name, role= "ADMIN", username=Username, email = email)
79             else:
80                 msg = "Incorrect Email/password"
81
82             return render_template("login.html", msg=msg)
83         else:
84             return render_template("login.html")
85

```

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

```
87 @app.route("/register", methods=['POST', 'GET'])
88 def signup():
89     msg = ''
90     if request.method == 'POST':
91         name = request.form["sname"]
92         email = request.form["semail"]
93         username = request.form["susername"]
94         role = int(request.form['role'])
95         password = ''.join(random.choice(string.ascii_letters) for i in range(0,8))
96         link = 'https://university.ac.in/portal'
97         print(password)
98         sql = "SELECT* FROM register WHERE email= ?"
99         stmt = ibm_db.prepare(conn, sql)
100        ibm_db.bind_param(stmt, 1, email)
101        ibm_db.execute(stmt)
102        account = ibm_db.fetch_assoc(stmt)
103        print(account)
104        if account:
105            msg = "Already Registered"
106            return render_template('adminregister.html', error=True, msg=msg)
107        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
108            msg = "Invalid Email Address!"
109        else:
110            insert_sql = "INSERT INTO register VALUES (?,?,?,?,?,?)"
111            prep_stmt = ibm_db.prepare(conn, insert_sql)
112            # this username & password is should be same as db-2 details & order also
113            ibm_db.bind_param(prep_stmt, 1, name)
114            ibm_db.bind_param(prep_stmt, 2, email)
115            ibm_db.bind_param(prep_stmt, 3, username)
116            ibm_db.bind_param(prep_stmt, 5, role)
```

```

117     ibm_db.bind_param(prepare_stmt, 4, password)
118     ibm_db.execute(prepare_stmt)
119     payload = {
120         "personalizations": [
121             {
122                 "to": [{"email": email}],
123                 "subject": "Student Account Details"
124             }
125         ],
126         "from": {"email": "<Enter Your Sender Email Id>"},
127         "content": [
128             {
129                 "type": "text/plain",
130                 "value": """Dear {} , \n Welcome to University,
131                 Here there the details to Login Into your student portal link : {} \n
132                 YOUR Username : {} \n PASSWORD : {} \n Thank you \n Sincerely\n
133                 Office of Admissions\n Mahidhar Institute of Technology \n
134                 E-Mail: admission@university.ac.in ; Website: www.university.ac.in"""
135             .format( name,link, username, password)
136         }
137     ]
138 }
139     headers = {
140         "content-type": "application/json",
141         "X-RapidAPI-Key": "<Enter your Rapid api - api key>",
142         "X-RapidAPI-Host": "rapidprod-sendgrid-v1.p.rapidapi.com"
143     }
144     response = requests.request("POST", url, json=payload, headers=headers)
145     print(response.text)
146     msg = "Registration Successful"
147     return render_template('adminregister.html', msg=msg)

```

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

SendGrid FREEMIUM Verified Official
By Sendgrid | Updated a year ago | Email

Endpoints About Tutorials Discussions Pricing

Welcome to SendGrid's Web API v3! This API is RESTful, fully featured, and easy to integrate with.

V1 (Current)

1 Test Endpoint

2

3

4

5

```

url = "https://rapidprod-sendgrid-v1.p.rapidapi.com/mail/send"

payload = {
    "personalizations": [
        {
            "to": [{"email": "john@example.com"}],
            "subject": "Hello, World!"
        }
    ],
    "from": {"email": "from_address@example.com"},
    "content": [
        {
            "type": "text/plain",
            "value": "Hello, world!"
        }
    ]
}

headers = {
    "content-type": "application/json",
    "X-RapidAPI-Key": "714250c24fmsh829f65c05932f01p1ad65cjsn43a39e64b3bb",
    "X-RapidAPI-Host": "rapidprod-sendgrid-v1.p.rapidapi.com"
}

response = requests.post(url, json=payload, headers=headers)

```

- 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

```

148 @app.route("/studentssubmit", methods=['POST','GET'])
149 def sassignment():
150     u = Username.strip()
151     subtime = []
152     ma = []
153     sql = "SELECT SUBMITTIME, MARKS from SUBMIT WHERE STUDENTNAME = ? "
154     stmt = ibm_db.prepare(conn, sql)
155     ibm_db.bind_param(stmt, 1, u)
156     ibm_db.execute(stmt)
157     st = ibm_db.fetch_tuple(stmt)
158     while st !=False:
159         subtime.append(st[0])
160         ma.append(st[1])
161         st = ibm_db.fetch_tuple(stmt)
162     print(subtime)
163     print(ma)
164     if request.method=="POST":
165         for x in range (1,5):
166             x = str(x)
167             y = str("file"+x)
168             print(type(y))
169             f=request.files[ y ]
170             print(f)
171             print(f.filename)
172
173
174
175     if f.filename != '':
176
177         basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
178         #print("current path",basepath)
179         filepath=os.path.join(basepath,'uploads',u+x+".pdf") #from anywhere in the system we can give image but we want that image l
180         #print("upload folder is",filepath)
181         f.save(filepath)

```

```

182
183     # connecting with cloud object storage
184
185     COS_ENDPOINT = "<Enter your COS EndPoint>"
186     COS_API_KEY_ID = "<Enter your COS API Key>"
187     COS_INSTANCE_CRN = "<Enter your COS CRN Instance id>"
188     cos = ibm_boto3.client("s3",ibm_api_key_id=COS_API_KEY_ID,ibm_service_instance_id=COS_INSTANCE_CRN,
189                           config=Config(signature_version="oauth"),endpoint_url=COS_ENDPOINT)
190     cos.upload_file(Filename=filepath,Bucket='studentassignment',Key= u+x+".pdf")
191     msg = "Uploading Successful"
192     ts = datetime.datetime.now()
193     t = ts.strftime("%Y-%m-%d %H:%M:%S")
194     sql1 = "SELECT * FROM SUBMIT WHERE STUDENTNAME = ? AND ASSIGNMENTNUM = ?"
195     stmt = ibm_db.prepare(conn, sql1)
196     ibm_db.bind_param(stmt, 1, u)
197     ibm_db.bind_param(stmt, 2, x)
198     ibm_db.execute(stmt)
199     acc = ibm_db.fetch_assoc(stmt)
200     print(acc)
201     if acc == False:
202         sql = "INSERT into SUBMIT (STUDENTNAME, ASSIGNMENTNUM, SUBMITTIME) values (?,?,?)"
203         stmt = ibm_db.prepare(conn, sql)
204         ibm_db.bind_param(stmt, 1, u)
205         ibm_db.bind_param(stmt, 2, x)
206         ibm_db.bind_param(stmt, 3, t)
207         ibm_db.execute(stmt)
208     else:
209         sql = "UPDATE SUBMIT SET SUBMITTIME = ? WHERE STUDENTNAME = ? and ASSIGNMENTNUM = ?"
210         stmt = ibm_db.prepare(conn, sql)
211         ibm_db.bind_param(stmt, 1, t)
212         ibm_db.bind_param(stmt, 2, u)
213         ibm_db.bind_param(stmt, 3, x)
214         ibm_db.execute(stmt)
215
216         return render_template("studentssubmit.html", msg=msg, datetime=subtime, marks=ma)
217         continue
218     return render_template("studentssubmit.html", datetime=subtime, Marks=ma)

```

3.11

Rendering Facultyprofile.Html

```
35     @app.route("/facultyprofile")
36     def fprofile():
37         return render_template("facultyprofile.html")
38
```

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, L=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

```
243     @app.route("/marksassign/<string:stdname>", methods=['POST', 'GET'])
244     def marksassign(stdname):
245         global u
246         global g
247         global file
248         da = []
249
250         COS_ENDPOINT = "Enter your COS Endpoint"
251         COS_API_KEY_ID = "Enter your COS api key"
252         COS_INSTANCE_CRN = "Enter your COS CRN ID"
253         cos = ibm_boto3.client("s3",
254                               ibm_api_key_id=COS_API_KEY_ID,
255                               ibm_service_instance_id=COS_INSTANCE_CRN,
256                               config=Config(signature_version="oauth"),
257                               endpoint_url=COS_ENDPOINT)
258         output = cos.list_objects(Bucket="studentassignmentsb")
259         output
260         l=[]
261         for i in range(0,len(output['Contents'])):
262             j = output['Contents'][i]['Key']
263             l.append(j)
264         l
265         u = stdname
266         print(len(u))
267         print(len(l))
268         n = []
269         for i in range(0,len(l)):
270             for j in range(0,len(u)):
271                 if u[j]==l[i][j]:
272                     n.append(l[i])
```

```
273     file = set(n)
274     file = list(file)
275     print(file)
276     print(len(file))
277     g = len(file)
278     sql = "SELECT SUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
279     stmt = ibm_db.prepare(conn, sql)
280     ibm_db.bind_param(stmt, 1, u)
281     ibm_db.execute(stmt)
282     m = ibm_db.fetch_tuple(stmt)
283     while m != False:
284         da.append(m[0])
285         m = ibm_db.fetch_tuple(stmt)
286
287     print(da)
288     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.

```
290 @app.route("/marksupdate/<string:anum>", methods=['POST', 'GET'])
291 def marksupdate(anum):
292     ma = []
293     da = []
294     mark = request.form['mark']
295     print(mark)
296     print(u)
297     sql = "UPDATE SUBMIT SET MARKS = ? WHERE STUDENTNAME = ? and ASSIGNMENTNUM = ?"
298     stmt = ibm_db.prepare(conn, sql)
299     ibm_db.bind_param(stmt, 1, mark)
300     ibm_db.bind_param(stmt, 2, u)
301     ibm_db.bind_param(stmt, 3, anum)
302     ibm_db.execute(stmt)
303     msg = "MARKS UPDATED"
304     sql = "SELECT MARKS, SUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
305     stmt = ibm_db.prepare(conn, sql)
306     ibm_db.bind_param(stmt, 1, u)
307     ibm_db.execute(stmt)
308     m = ibm_db.fetch_tuple(stmt)
309     while m != False:
310         ma.append(m[0])
311         da.append(m[1])
312         m = ibm_db.fetch_tuple(stmt)
313         #ma = ma[0]
314     print(ma)
315     print(da)
316     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.

```

322     @app.route("/logout")
323     def logout():
324         session.pop('loggedin', None)
325         session.pop('id', None)
326         session.pop('username', None)
327         return render_template("logout.html")

```

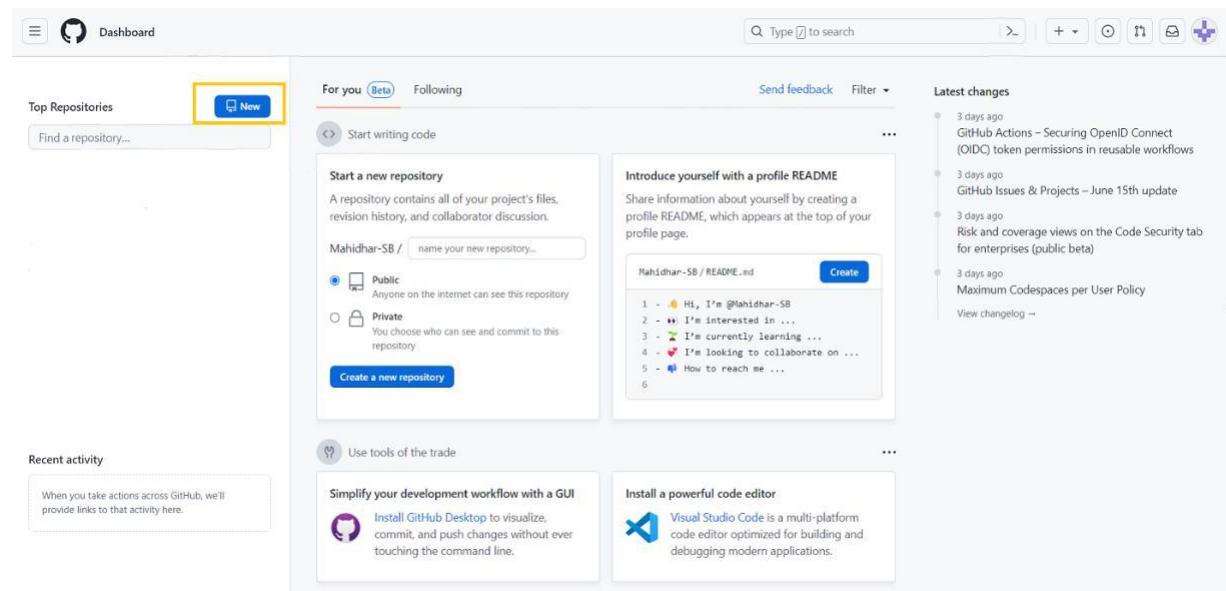
Here we will remove the saved email id and username in session, because now the user is not active so, no need to track.

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.



The screenshot shows the GitHub dashboard. At the top, there's a navigation bar with 'Dashboard' and other icons. Below it, a search bar says 'Type / to search'. On the left, there's a sidebar with 'Top Repositories' and a 'New' button highlighted with a yellow box. A 'Find a repository...' input field is also visible. The main area has several sections: 'For you Beta' (with a 'Start writing code' button), 'Following' (empty), 'Send feedback' and 'Filter' dropdowns, and a 'Latest changes' sidebar. The 'Latest changes' sidebar lists recent updates from Mahidhar-SB, such as 'Actions – Securing OpenID Connect (OIDC) token permissions in reusable workflows' and 'Issues & Projects – June 15th update'. At the bottom, there are sections for 'Recent activity', 'Use tools of the trade', 'Simplify your development workflow with a GUI', and 'Install a powerful code editor'.

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

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * Mahidhar-SB / Repository name * studentassignment / studentassignment is available.

Great repository names are short and memorable. Need inspiration? How about probable-goggles?

Description (optional)

Public Anyone on the internet can see this repository. You choose who can commit.
 Private You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore .gitignore template: None Choose which files not to track from a list of templates. Learn more about ignoring files.

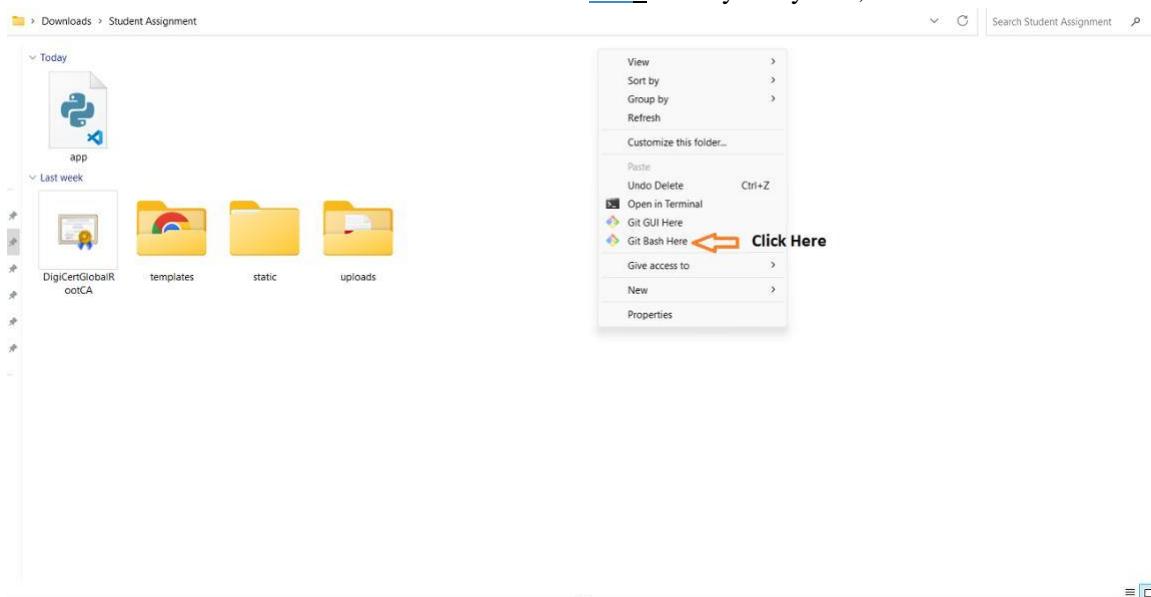
Choose a license License: None A license tells others what they can and can't do with your code. Learn more about licenses.

You are creating a public repository in your personal account.

Create repository

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.

The screenshot shows a GitHub repository page for 'studentassignment' (Public). At the top, there are buttons for Pin, Unwatch, Fork, Star, and more. Below the header, there are two sections: 'Set up GitHub Copilot' (using GitHub's AI pair programmer) and 'Invite collaborators' (finding people by GitHub username or email address). A large blue box titled 'Quick setup — if you've done this kind of thing before' contains three sections: '...or create a new repository on the command line' with a code snippet, '...or push an existing repository from the command line' with another code snippet, and '...or import code from another repository'.

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.

The screenshot shows a GitHub repository page for 'studentassignment'. The repository has 1 star, 1 watch, and 0 forks. The commit history shows a single commit from 'Mahidhar-SB' titled 'first commit' made 8 minutes ago. The commit includes files: static, templates, uploads, DigiCertGlobalRootCA.crt, and app.py. The repository also has sections for About, Releases, Packages, and Languages.

5

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.

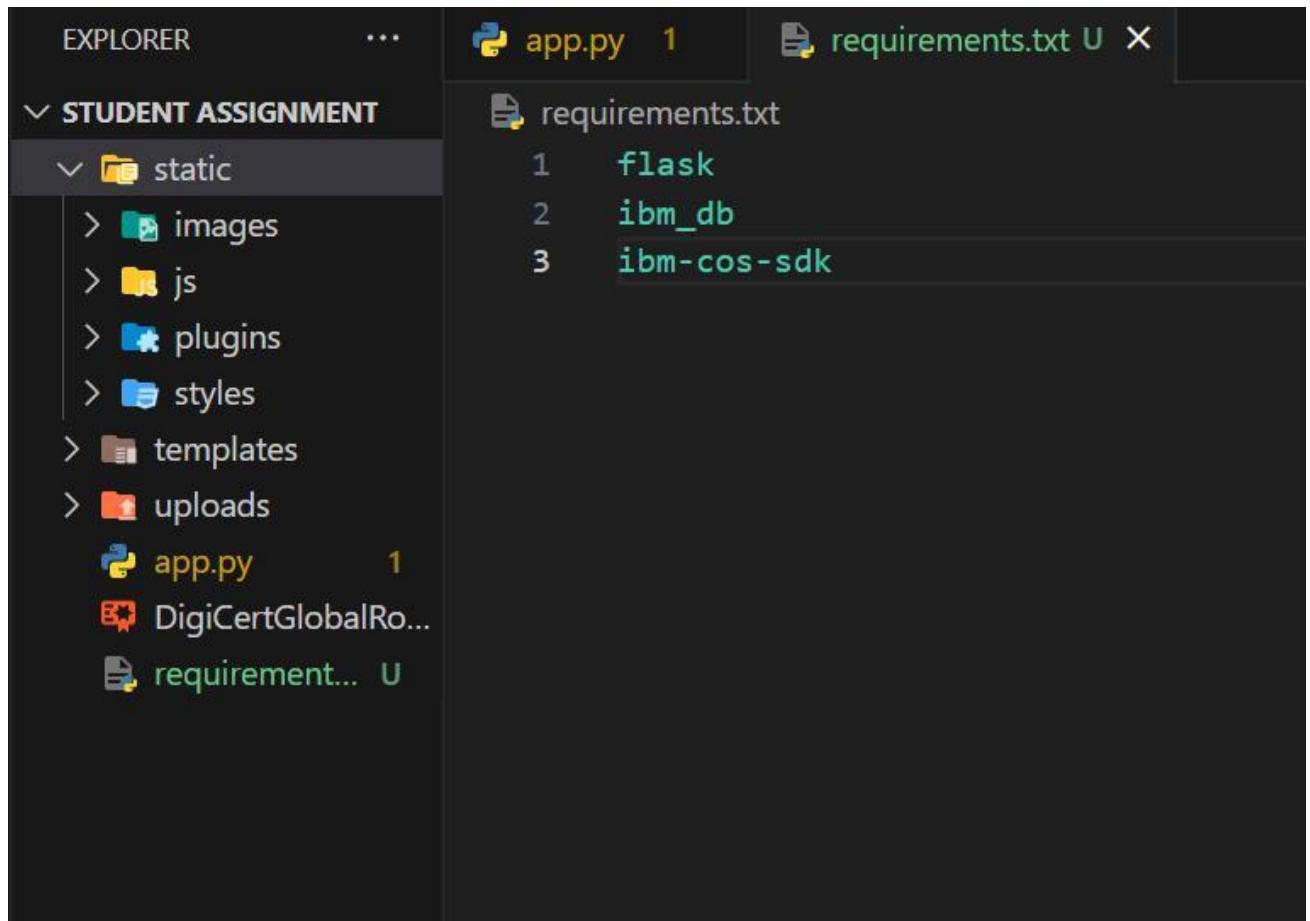
5.1

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.

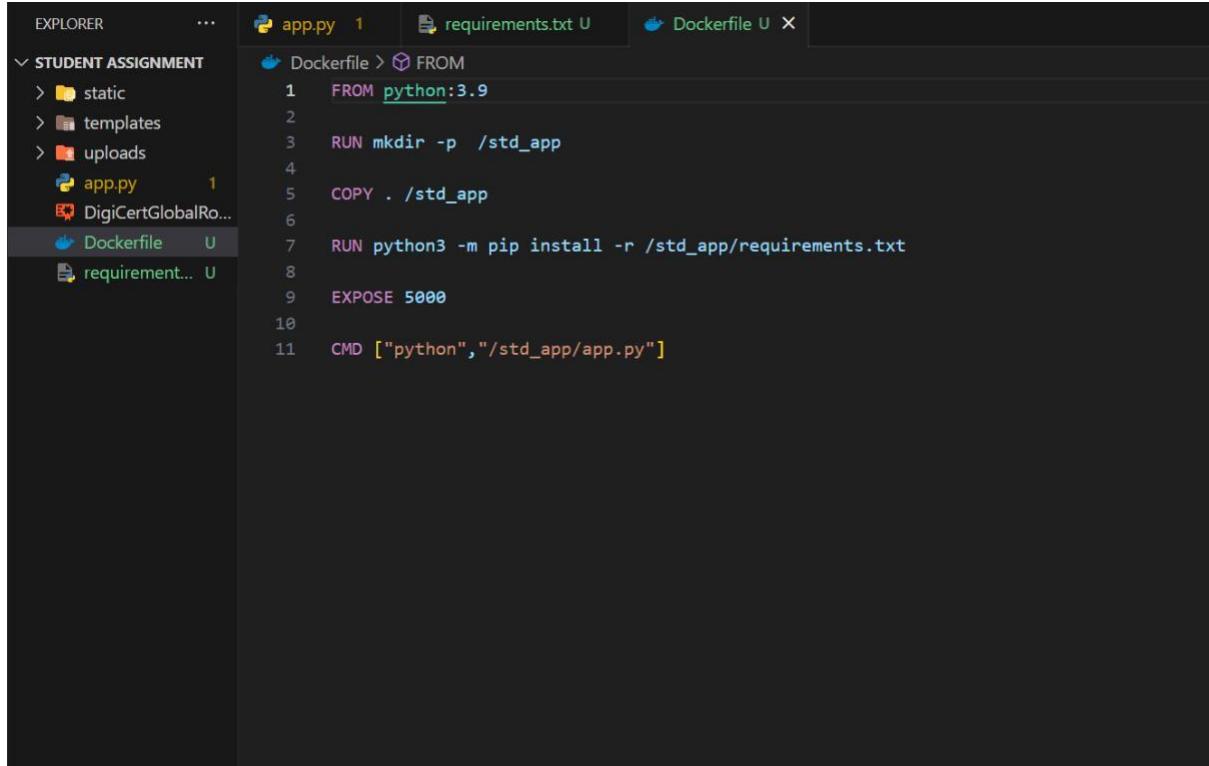


```
requirements.txt
1 flask
2 ibm_db
3 ibm-cos-sdk
```

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.



The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar showing a project structure under 'STUDENT ASSIGNMENT'. The files listed are static, templates, uploads, app.py (1), DigiCertGlobalRo..., Dockerfile (U), and requirements.txt. The Dockerfile tab is selected, showing the following content:

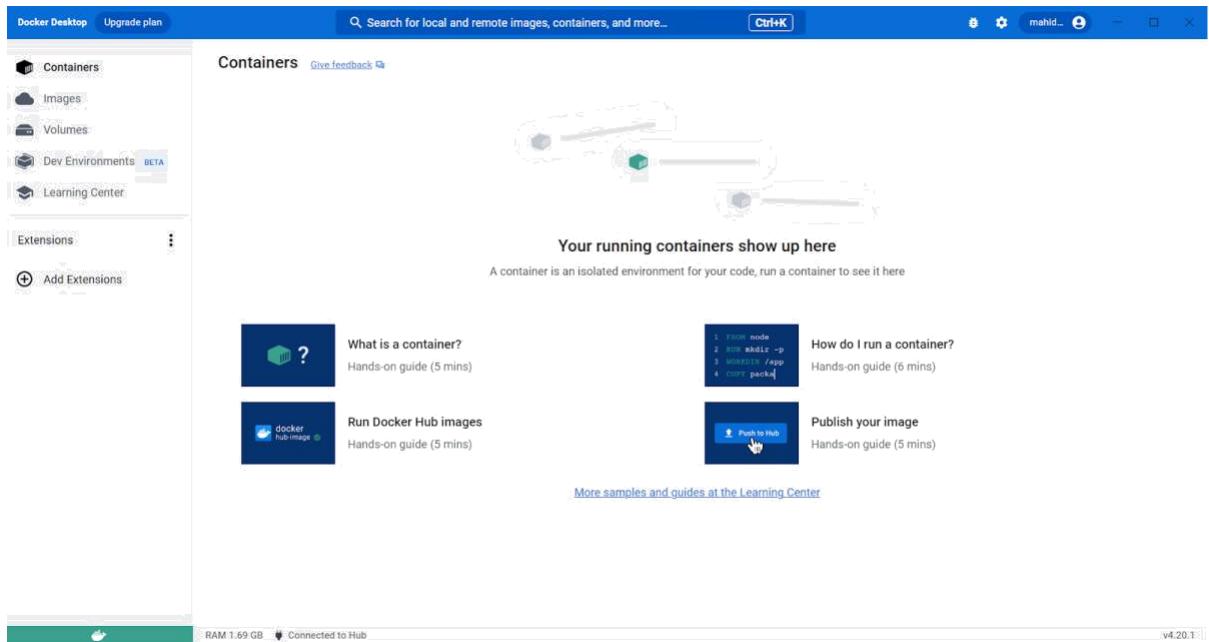
```
FROM python:3.9
RUN mkdir -p /std_app
COPY . /std_app
RUN python3 -m pip install -r /std_app/requirements.txt
EXPOSE 5000
CMD ["python", "/std_app/app.py"]
```

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.

```
C:\Windows\System32\cmd.exe + ^
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\smart\Downloads\Student Assignment>docker version
Client:
  Cloud integration: v1.0.33
  Version:          24.0.2
  API version:      1.43
  Go version:       go1.20.4
  Git commit:       cb74dfc
  Built:            Thu May 25 21:53:15 2023
  OS/Arch:          windows/amd64
  Context:          default

Server: Docker Desktop 4.20.1 (110738)
Engine:
  Version:          24.0.2
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.4
  Git commit:       659604f
  Built:            Thu May 25 21:52:17 2023
  OS/Arch:          linux/amd64
  Experimental:    false
containerd:
  Version:          1.6.21
  GitCommit:        3dce8eb055cbb6872793272b4f20ed16117344f8
runc:
  Version:          1.1.7
  GitCommit:        v1.1.7-0-g860f061
dockerd-init:
  Version:          0.19.0
  GitCommit:        de40ad0

C:\Users\smart\Downloads\Student Assignment>
```

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 989.7s (10/10) FINISHED
=> [internal] load .dockerignore
=> [internal] load context: 2B
=> [internal] load build definition from Dockerfile
=> [internal] transfering dockerfile: 209B
=> [internal] load metadata for docker.io/library/python:3.9
=> [auth] library/python:3 pull token for registry-1.docker.io
=> [internal] load build context
=> transferring context: 11.68MB
[1/4] FROM docker.io/library/python:3.9-sha256:98f018a1af6d7f2e17a0abd5bf89b998734ba7clee54780e7ed216f8b8095c3
=> resolve docker.io/library/python:3.9-sha256:98f018a1af6d7f2e17a0abd5bf89b998734ba7clee54780e7ed216f8b8095c3
=> sha256:98f018a1af6d7f2e17a0abd5bf89b998734ba7clee474878e7ed216f8b8095c3 1.86KB / 1.86KB
=> sha256:d56fdccf73c186e83b9933d499461b75bd2eb5beb0d996758bed81fbef5a22d2 2.01KB / 2.01KB
=> sha256:418db5cc0a24c7db9c2279147b637b8906fcfb23d1laadde5de1f70a153c6807 7.51KB / 7.51KB
=> sha256:24ff2345db06550282f6ge0a46f111fb2d5dcfc67e871f89b509151bf6 64.11MB / 64.11MB
=> sha256:b2110d5bcaad06ab3ac3bf7d37390c460b2a688529b1d118a79991116f4 49.55MB / 49.55MB
=> sha256:c2b280be87758dde67c79b25dulcbf88377601a3255cc5d55569abeebe80da00 24.03MB / 24.03MB
=> sha256:7c62c9249808a64728effb8f1249dc5585b412b1a358bbas668502e4411d2667639 211.00MB / 211.00MB
=> sha256:bba7bb10d5baeacaad06ab3ac3bf7d37390c460b2a688529b1d118a79991116f4
=> extracting sha256:bba7bb10d5baeacaad06ab3ac3bf7d37390c460b2a688529b1d118a79991116f4
=> extracting sha256:b210932934efec1c984a1fad98942bb3f4d73106610edbd5185fc037f18409 15.82MB / 15.82MB
=> extracting sha256:7758de6779b25dulcbf88377601a3255cc5d5569abeebe80da00
=> sha256:b2210932934efec1c984a1fad98942bb3f4d73106610edbd5185fc037f18409 15.82MB
=> extracting sha256:7758de6779b25dulcbf88377601a3255cc5d5569abeebe80da00
=> sha256:e9c01829d92f672ccecca7ea47fc437ad8ed4cal4f62101f76d22881bf54231e 242B / 242B
=> sha256:d628541f1b2d887758dde67c79b25dulcbf88377601a3255cc5d5569abeebe80da00
=> extracting sha256:fe23129f080a0e28effb8f1249dc5585b412b1a358bbas668502e4411d2667639
=> extracting sha256:7c62c9249808a64728effb8f1249dc5585b412b1a358bbas668502e4411d2667639
=> extracting sha256:b2110932934efec1c984a1fad98942bb3f4d73106610edbd5185fc037f18409
=> extracting sha256:e9c01829d92f672ccecca7ea47fc437ad8ed4cal4f62101f76d22881bf54231e
=> extracting sha256:d628541f1b2d887758dde67c79b25dulcbf88377601a3255cc5d5569abeebe80da00
=> [2/4] RUN mkdir -p /std_app
=> [3/4] COPY . /std_app
=> [4/4] RUN python3 -m pip install -r /std_app/requirements.txt
=> exporting to image
=> exporting layers
=> writing image sha256:f0cf2d9762a7f044cf2f2141f8cd5e9a7135d0b4775b1c4344ecb9d5cd5a4b89
=> naming to docker.io/library/studentassignment:1.0
```

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 X

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

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: std-app-deploy
5    labels:
6      app: std-app
7  spec:
8    replicas: 3
9    selector:
10      matchLabels:
11        app: std-app
12    template:
13      metadata:
14        labels:
15          app: std-app
16      spec:
17        containers:
18          - name: std-app
19            image: mahidhar23/studentassignment:1.0
20            ports:
21              - containerPort: 5000
22
```

The screenshot shows a code editor interface with two tabs open: `deployment.yaml` and `service.yaml`. The `service.yaml` tab is active, displaying the following YAML configuration:

```
apiVersion: v1
kind: Service
metadata:
  name: std-service
spec:
  selector:
    app: std-app
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
      nodePort: 32123
```

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

8

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..

The screenshot shows the Red Hat Developer website with a dark theme. At the top, there's a navigation bar with links for "Start building apps", "Products & technologies", "Events", "Learn", "Developer Sandbox", "DevNation", and "Blog". On the far right are search and account icons. Below the navigation, a banner says "Learn containers, Kubernetes, and OpenShift in your browser". A main heading "Start exploring in the Developer Sandbox for free" is followed by a sub-instruction: "Try Red Hat's products and technologies without setup or configuration." A call-to-action button "Start your sandbox for free" is highlighted with an orange border. To the right, there's an illustration of a laptop displaying a terminal window with system specifications: "RAM: 7GB", "Storage: 150B", "Time limit: 30 days", and "Awesome: YES". Below the main content is a link "Get started with Red Hat OpenShift Service on AWS".

[Get started in the Developer Sandbox](#) [Developer Sandbox IDE](#) [Use case activities](#) [FAQs](#)



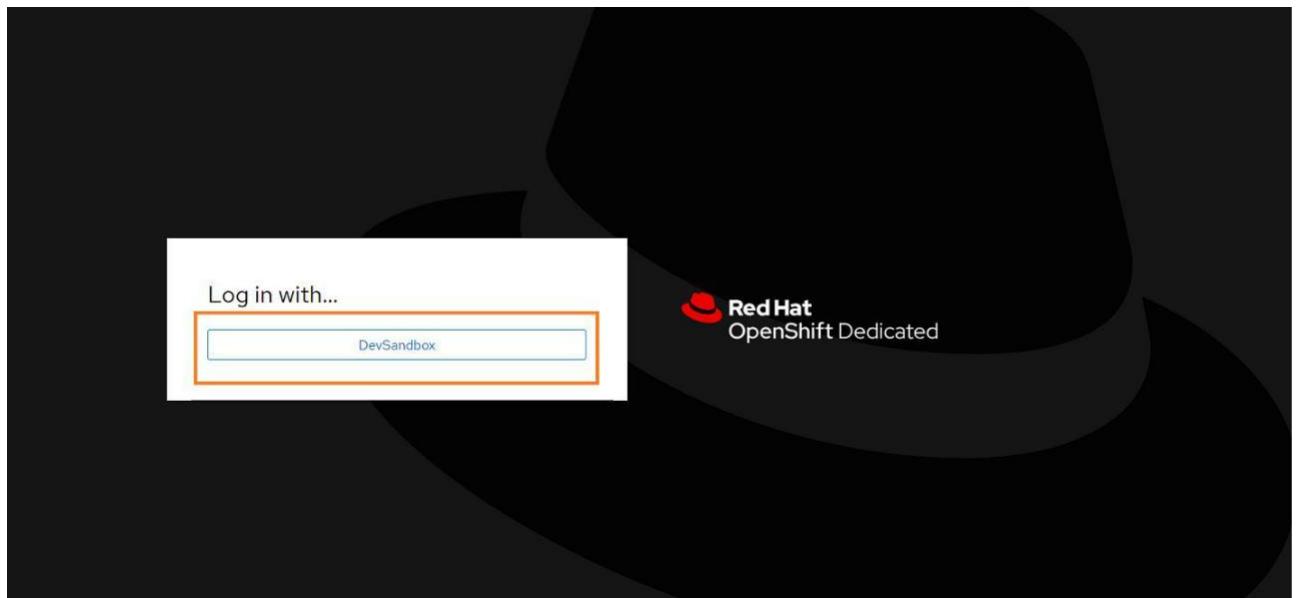
What is the Developer Sandbox?

Get 30-days free access to a shared OpenShift and Kubernetes cluster.

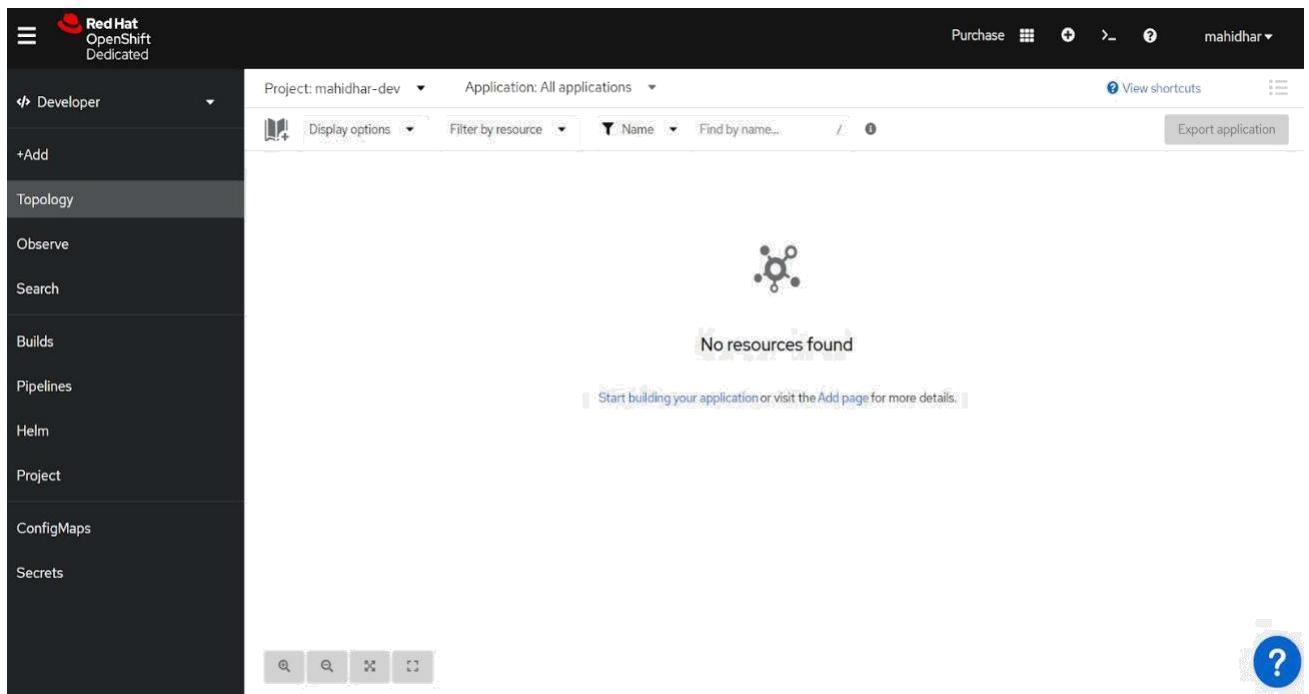
1 Guided tutorials to experience and run sample applications



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

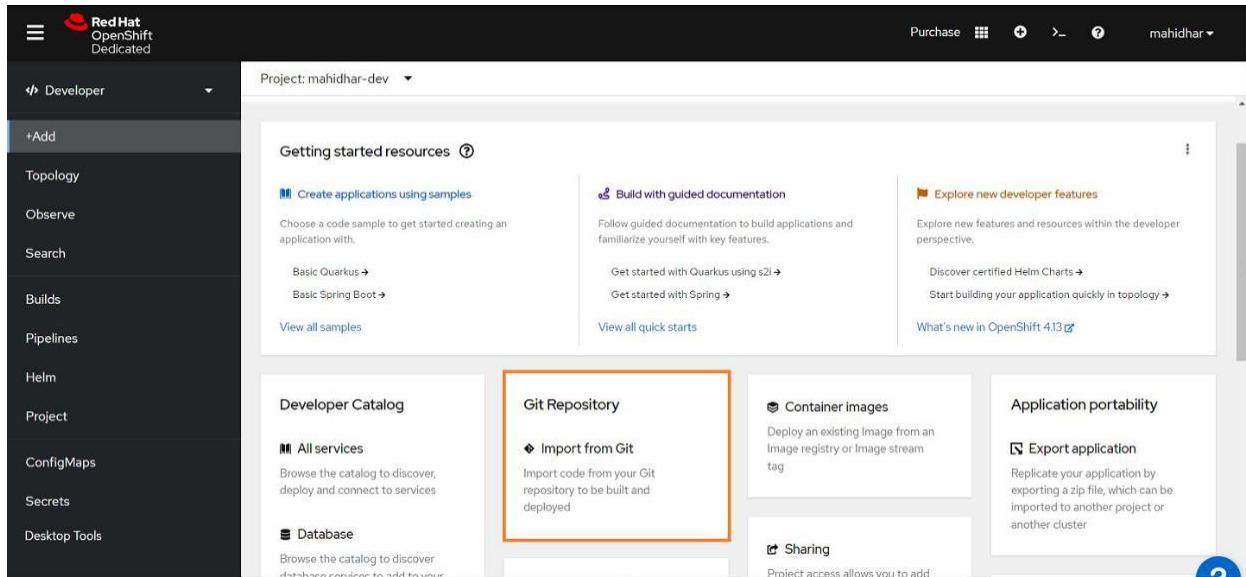


Stay on this page as developer and now click on that +add button on the left side panel. Here we can deploy our application in three ways.

9.

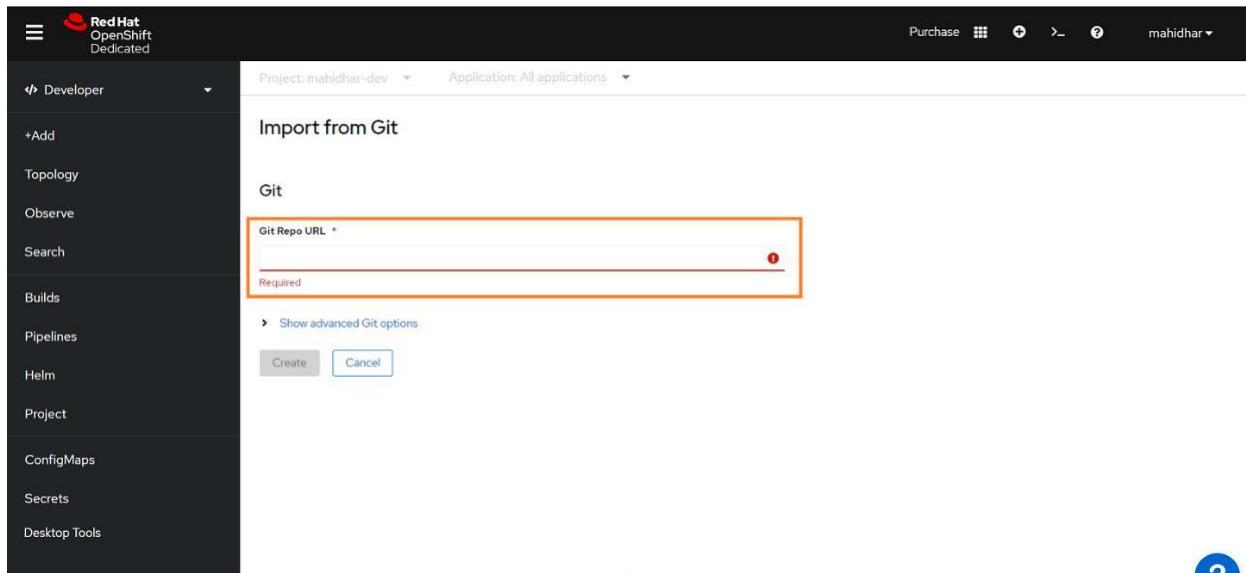
Deploying Application In Openshift Using Github

Click on that import from git option



The screenshot shows the Red Hat OpenShift Dedicated web interface. On the left, there's a sidebar with various project management options like Topology, Observes, Search, Builds, Pipelines, Helm, Project, ConfigMaps, Secrets, and Desktop Tools. The main area is titled 'Getting started resources'. It includes sections for 'Create applications using samples', 'Build with guided documentation', and 'Explore new developer features'. A central box is titled 'Developer Catalog' and contains 'All services' and 'Database' options. To the right, there's a larger box titled 'Git Repository' which contains the 'Import from Git' option, also highlighted with a red box. Other sections include 'Container images', 'Sharing', and 'Application portability'.

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



This screenshot shows the 'Import from Git' dialog box. The left sidebar remains the same as the previous screenshot. The main dialog has a title 'Import from Git' and a sub-section 'Git'. It features a large input field labeled 'Git Repo URL *' with a red border around it, indicating it's a required field. Below the input field is a small note 'Required'. There's also a link 'Show advanced Git options'. At the bottom are two buttons: 'Create' and 'Cancel'.

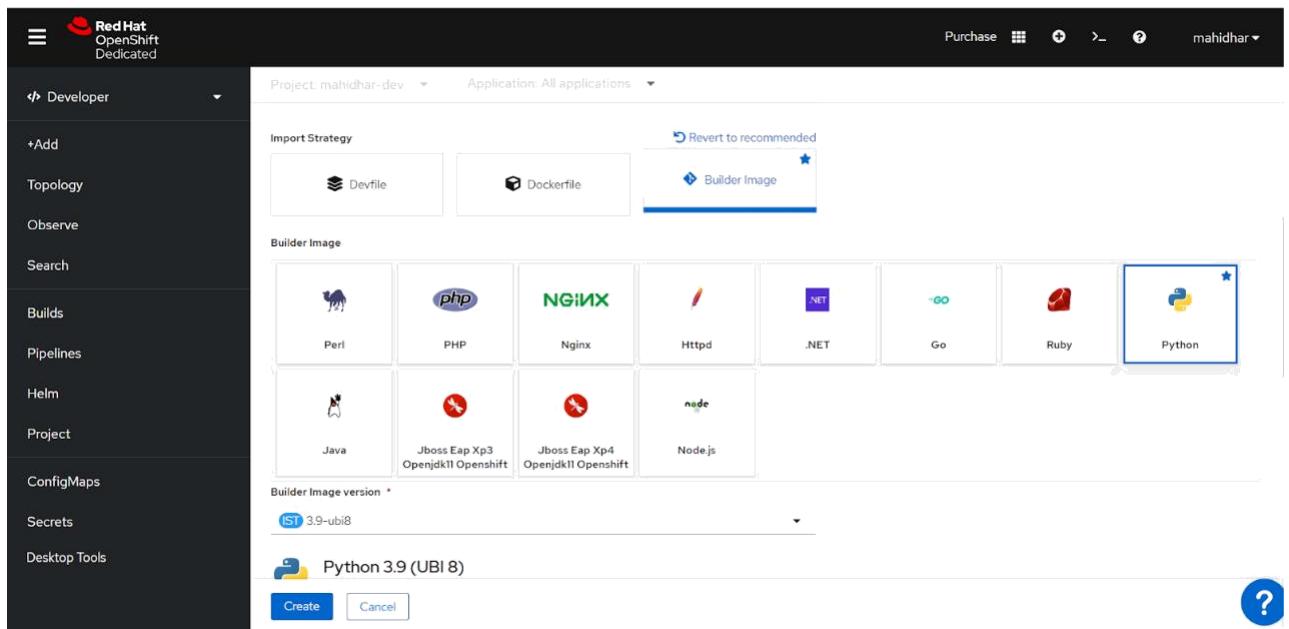
Go to github.com, login with your account and in repositories click on your project repository and copy the url of that repository.

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

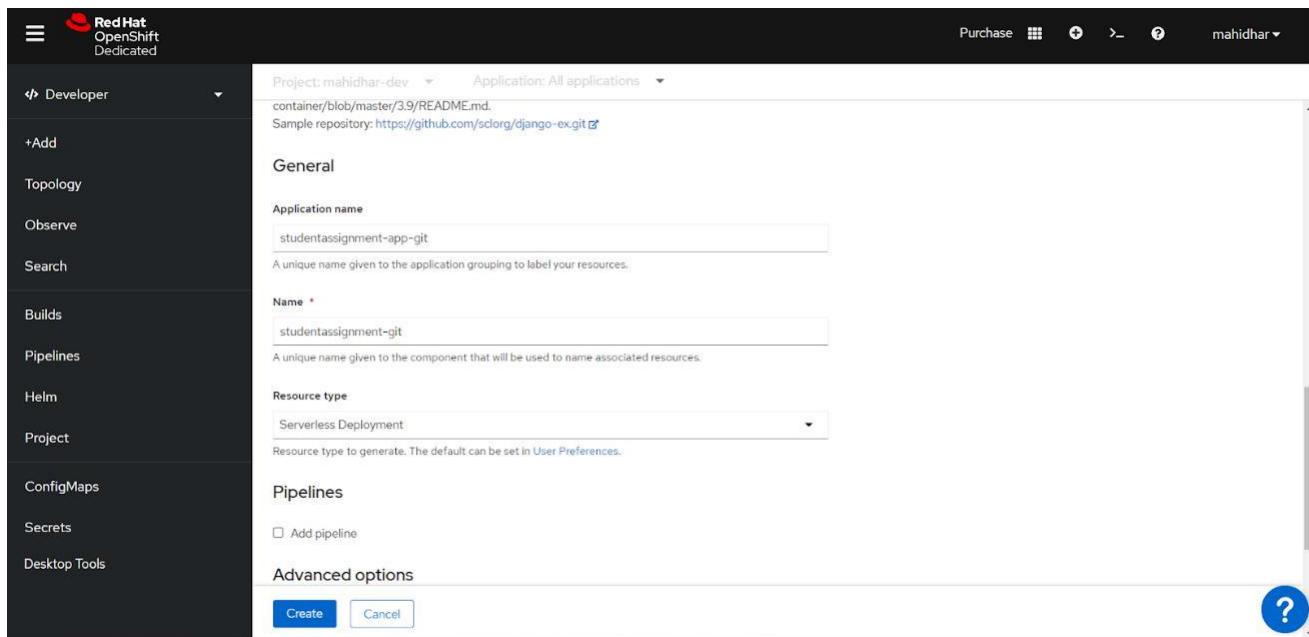
Once you paste it, validate and give us a green tick mark that path is correct. Now it will open a lot of options.

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

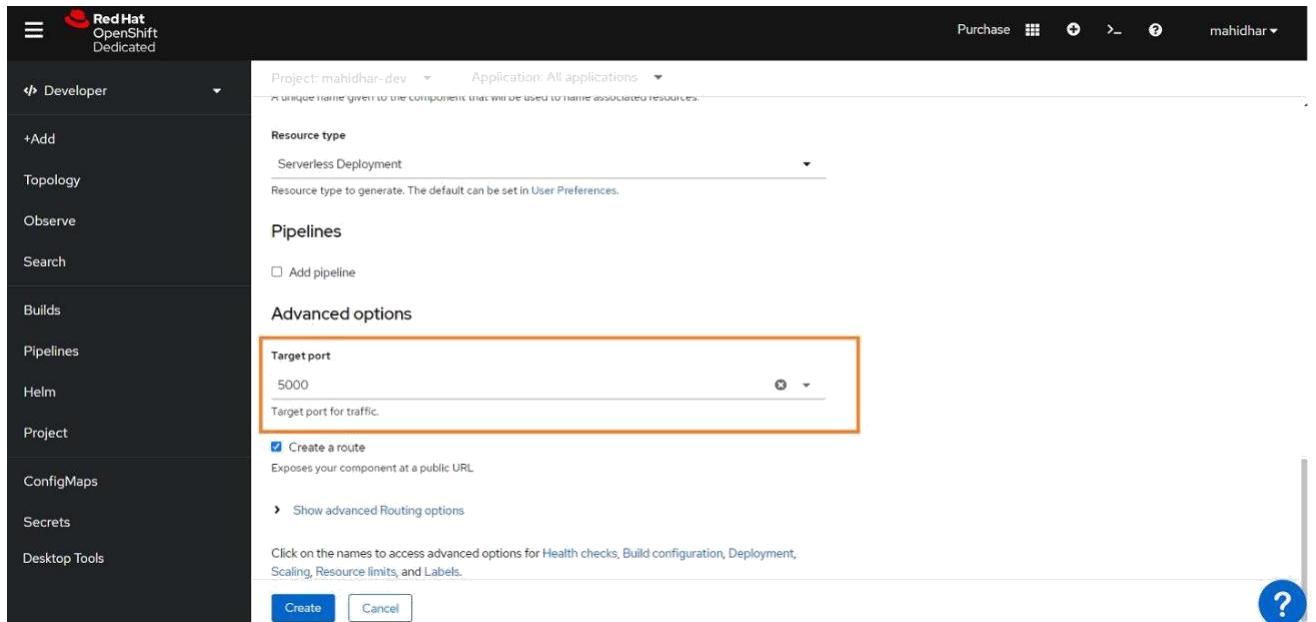


Now select python, as we used python programming language.

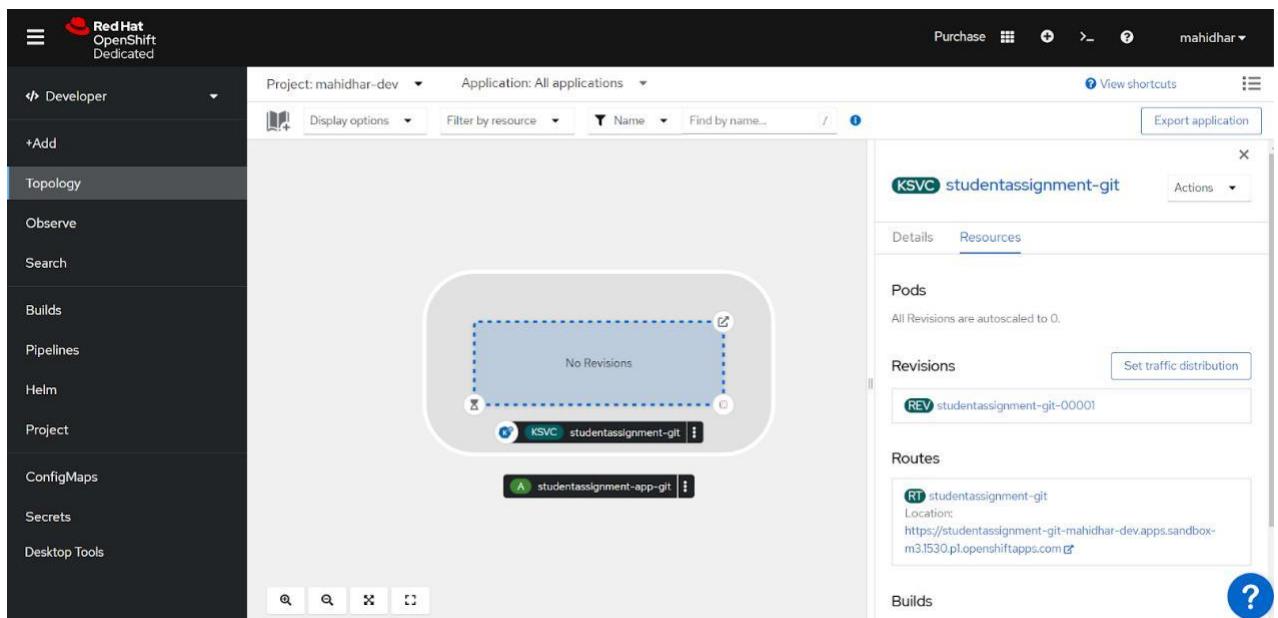
Now again scroll a little bit down, where in general section we can give some name for the application



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.

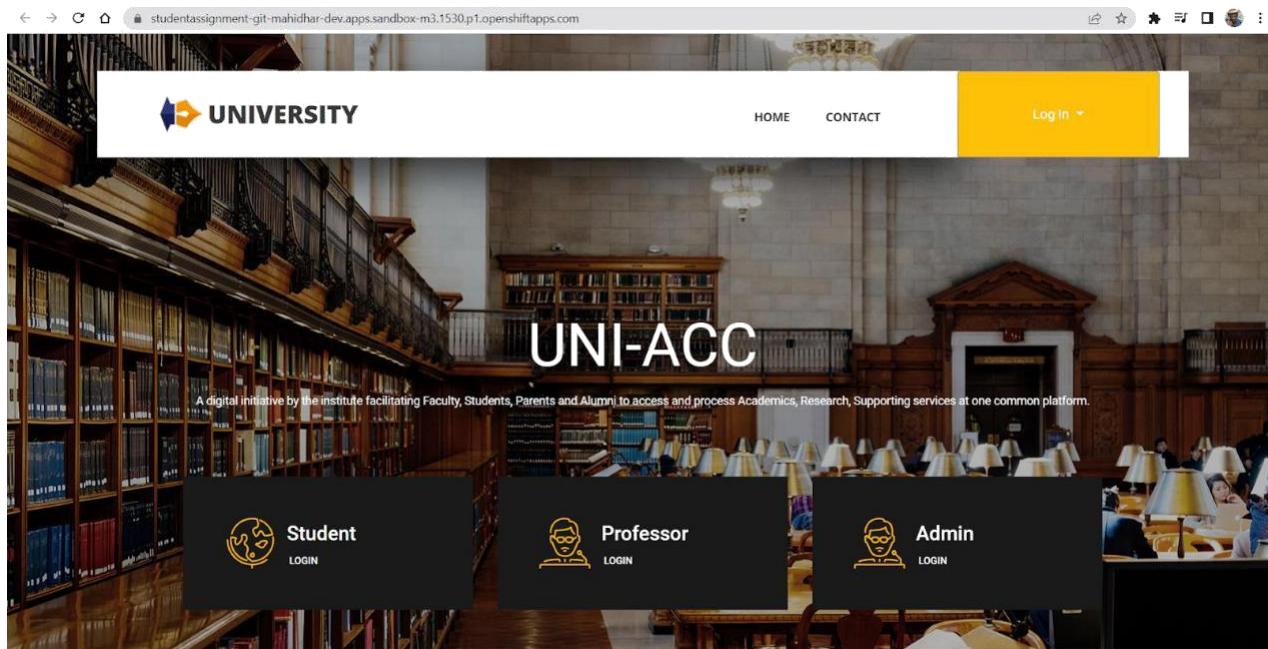


Once it's done, build the application the application will change like this.

The screenshot shows the Red Hat OpenShift Dedicated web interface. On the left, a sidebar menu includes options like Developer, Topology (which is selected), Observe, Search, Builds, Pipelines, Helm, Project, ConfigMaps, Secrets, and Desktop Tools. The main content area displays a pod diagram for the project 'mahidhar-dev'. The pod contains a Python icon and two containers: 'REV studentassignment-git-00001' and 'KSCV studentassignment-git'. A green arrow points from the Python icon to the 'REV' container. Below the pod is a button labeled 'studentassignment-app-git'. The right side of the screen shows tabs for 'Details' and 'Resources'. Under 'Pods', it says 'All Revisions are autoscaled to 0.' Under 'Revisions', there is one entry: 'REV studentassignment-git-00001' with '100%' traffic distribution, associated with a 'deployment' resource. Under 'Routes', there is one entry: 'RT studentassignment-git' with the location 'https://studentassignment-git-mahidhar-dev.apps.sandbox-m3.1530.p1.openshiftapps.com'. Under 'Builds', there is a blue circular icon with a question mark.

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

This screenshot is identical to the one above, showing the Red Hat OpenShift Dedicated interface. However, a white callout box with an orange border highlights the green arrow pointing from the Python icon to the 'REV' container in the pod diagram. This indicates where the user should click to publicly access the 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.

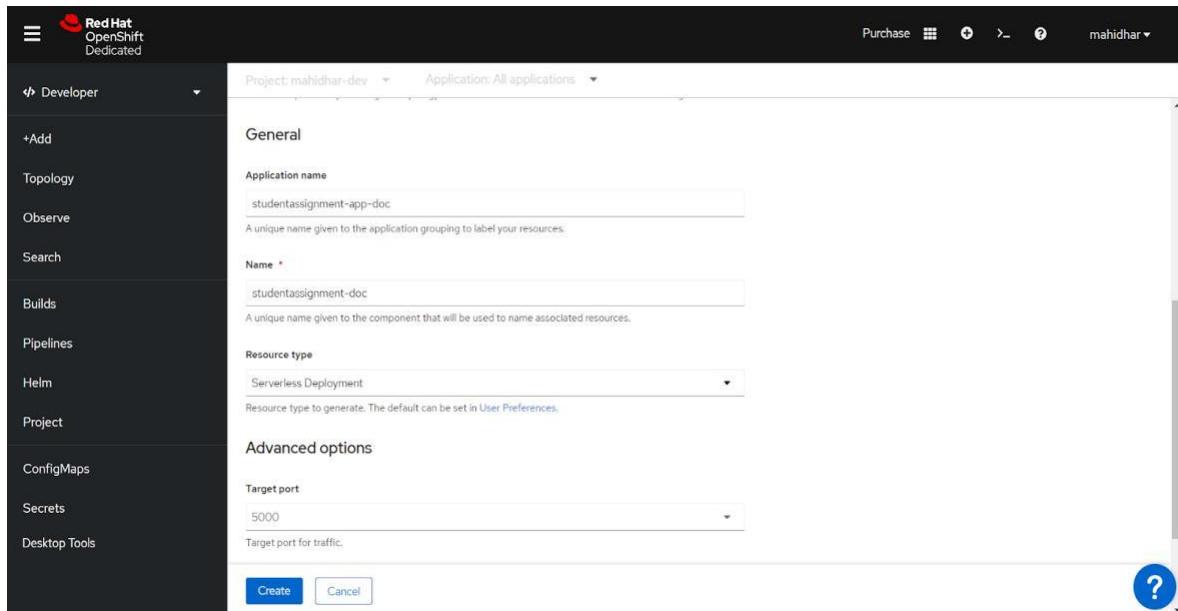
The screenshot shows the Red Hat OpenShift Dedicated developer interface. On the left, there's a sidebar with various options like Developer, Topology, Observe, Search, Builds, Pipelines, Helm, Project, ConfigMaps, Secrets, and Desktop Tools. The 'Developer' option is selected. The main area has a 'Project: mahidhar-dev' dropdown. Below it are sections for 'Basic Quarkus', 'Basic Spring Boot', 'View all samples', 'Get started with Quarkus using s2i', 'Get started with Spring', 'View all quick starts', 'Discover certified Helm Charts', 'Start building your application quickly in topology', and 'What's new in OpenShift 4.13'. A large central area is titled 'Developer Catalog' and contains sections for 'All services', 'Database', 'Operator Backed', and 'Helm Chart'. To the right, there are several boxes: 'Git Repository' (with 'Import from Git'), 'Container Images' (which is highlighted with an orange border), 'Sharing', 'From Local Machine', 'Pipelines', 'Application portability' (with 'Export application'), 'Eventing', 'Event Source', and 'Broker'. A question mark icon is in the bottom right corner.

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.

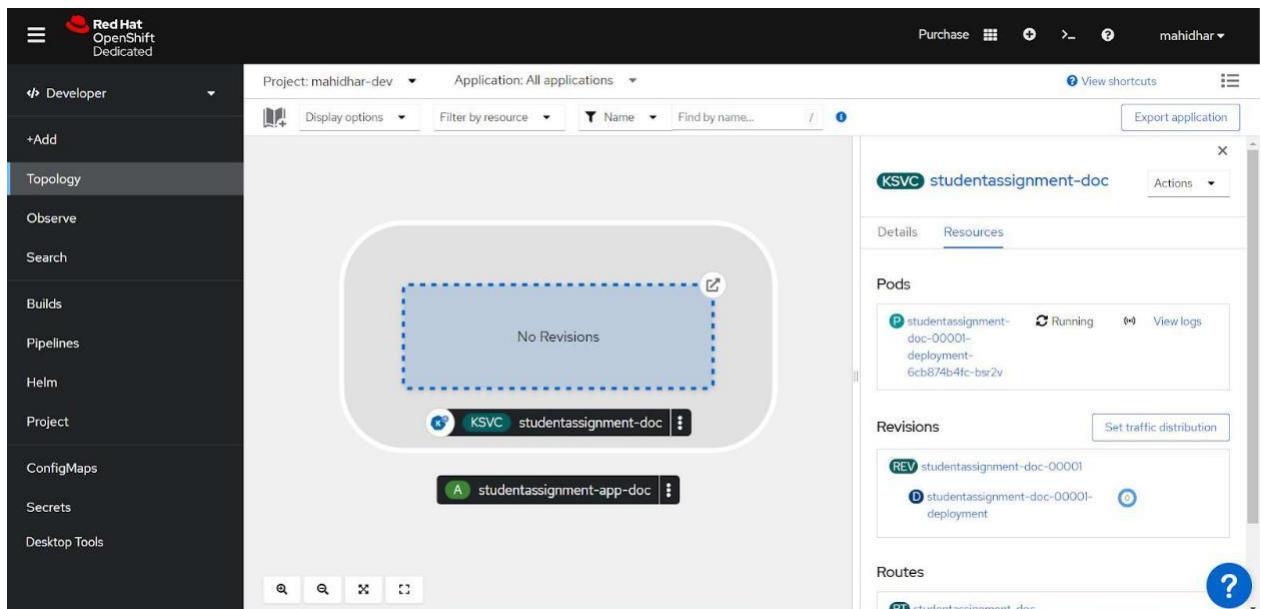
The screenshot shows the 'Deploy Image' configuration page. The left sidebar is identical to the previous one. The main area has a 'Deploy Image' title. Under the 'Image' section, it says 'Deploy an existing Image from an Image Stream or Image registry.' There are two radio buttons: 'Image name from external registry' (selected) and 'Image stream tag from internal registry'. The 'External Registry' field contains 'mahidhar23/studentassignment:1.0'. A note below says 'To deploy an Image from a private registry, you must create an Image pull secret with your Image registry credentials.' A checked checkbox 'Allow Images from insecure registries' is present. Under the 'Runtime icon' section, there's a dropdown menu with 'openshift' selected. A note below says 'The icon represents your Image in Topology view. A label will also be added to the resource defining the icon.' At the bottom, there are 'Create' and 'Cancel' buttons.

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.

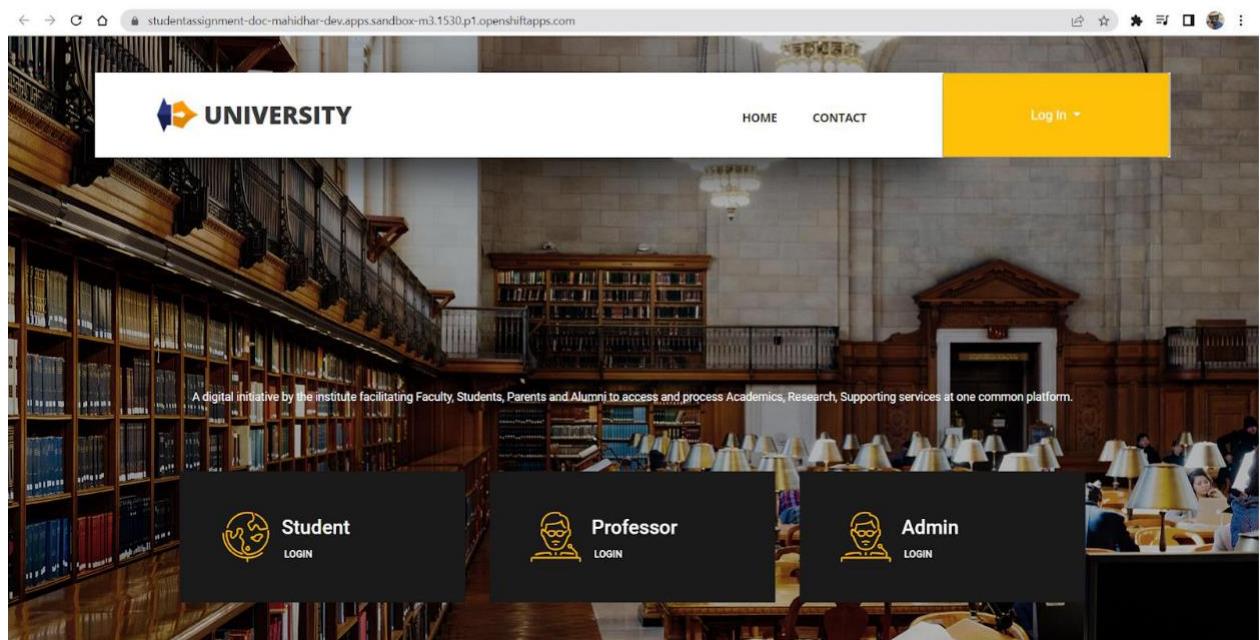
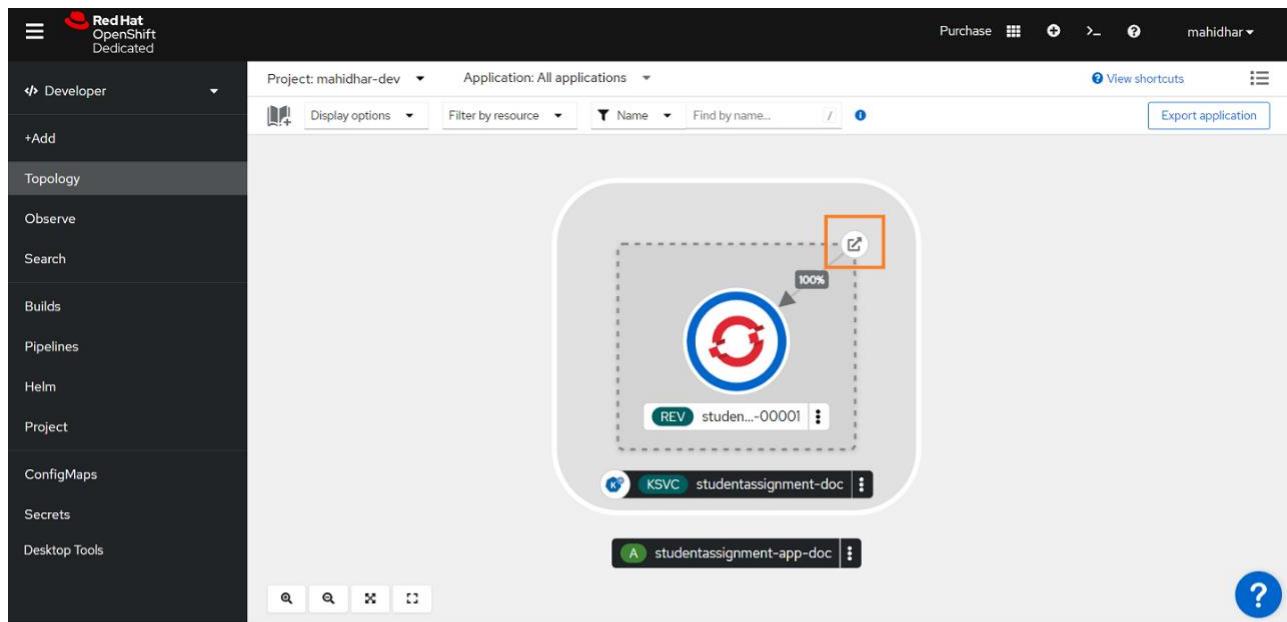
Now the same thing as we did above, name the application in the general section and port address as 5000 (or same port address of the flask) in the advanced options section.



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.



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.

The screenshot shows the Red Hat OpenShift Dedicated developer interface. On the left sidebar, under the '+Add' section, there are various options like Topology, Observe, Search, Builds, Pipelines, Helm, Project, ConfigMaps, Secrets, and Desktop Tools. The main content area is titled 'Project: mahidhar-dev'. It features several cards: 'Developer Catalog' (All services, Database, Operator Backed, Helm Chart), 'Git Repository' (Import from Git), 'Container images' (Deploy an existing Image from an Image registry or Image stream tag), 'Sharing' (Project access allows you to add or remove a user's access to the project), 'Application portability' (Export application, Eventing, Event Source, Broker), and 'From Local Machine' (Import YAML, Upload JAR file). The 'Import YAML' card is highlighted with an orange border.

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

The screenshot shows the 'Import YAML' dialog box. The left sidebar is identical to the previous one. The main area has a title 'Import YAML' with the instruction 'Drag and drop YAML or JSON files into the editor, or manually enter files and use --- to separate each definition.' Below this is a code editor containing the following YAML code:

```
12   Template:
13     Metadata:
14       Labels:
15         App: std-app
16     Spec:
17       Containers:
18         - Name: std-app
19           Image: mahidhar23/studentassignment:1.0
20         Ports:
21           - ContainerPort: 5000
22       ...
23
24
25   ApiVersion: v1
26   Kind: Service
27   Metadata:
28     Name: std-service
29   Spec:
30     Selector:
31       App: std-app
32     Type: LoadBalancer
```

At the bottom of the dialog are two buttons: 'Create' (highlighted with an orange border) and 'Cancel'.

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.

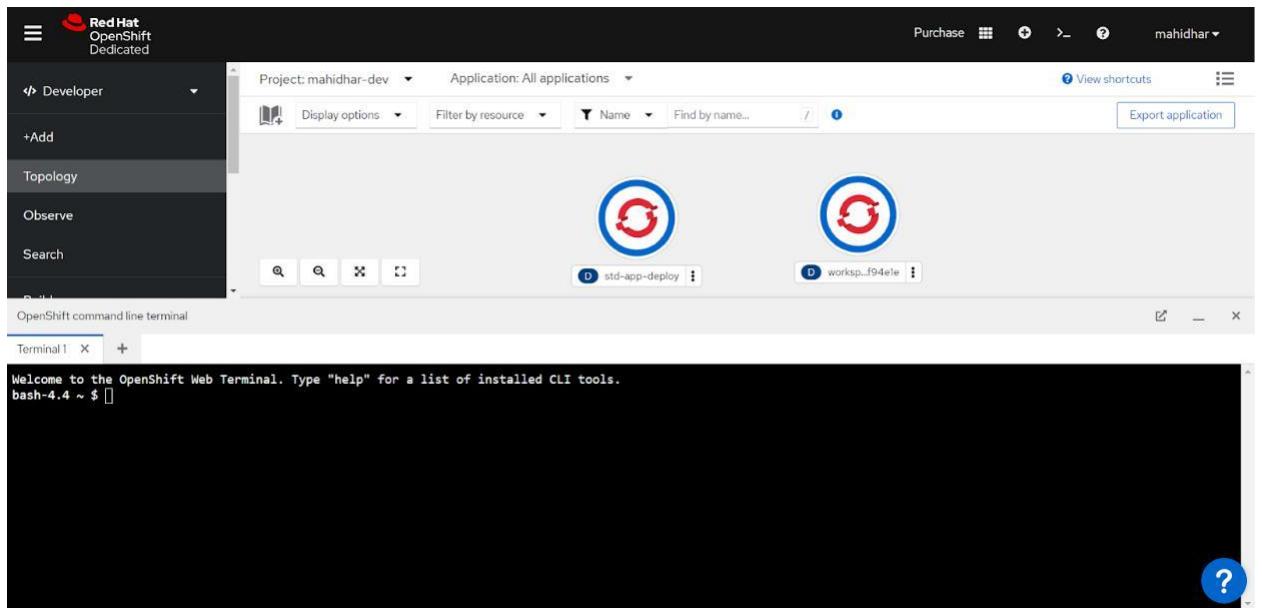
The screenshot shows the Red Hat OpenShift Dedicated web interface. On the left, a sidebar menu includes options like Developer, Topology, Observe, Search, Builds, Pipelines, Helm, Project, ConfigMaps, Secrets, and Desktop Tools. The main area displays a message "Resources successfully created" with a green checkmark icon. Below this, a table lists two resources: "std-app-deploy" and "std-service". Both entries show "Namespace: mahidhar-dev" and "Creation status: Created". A link "Import more YAML..." is visible at the bottom. The top right corner shows user information "mahidhar" and navigation icons. A blue question mark icon is in the bottom right corner of the main content area.

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.

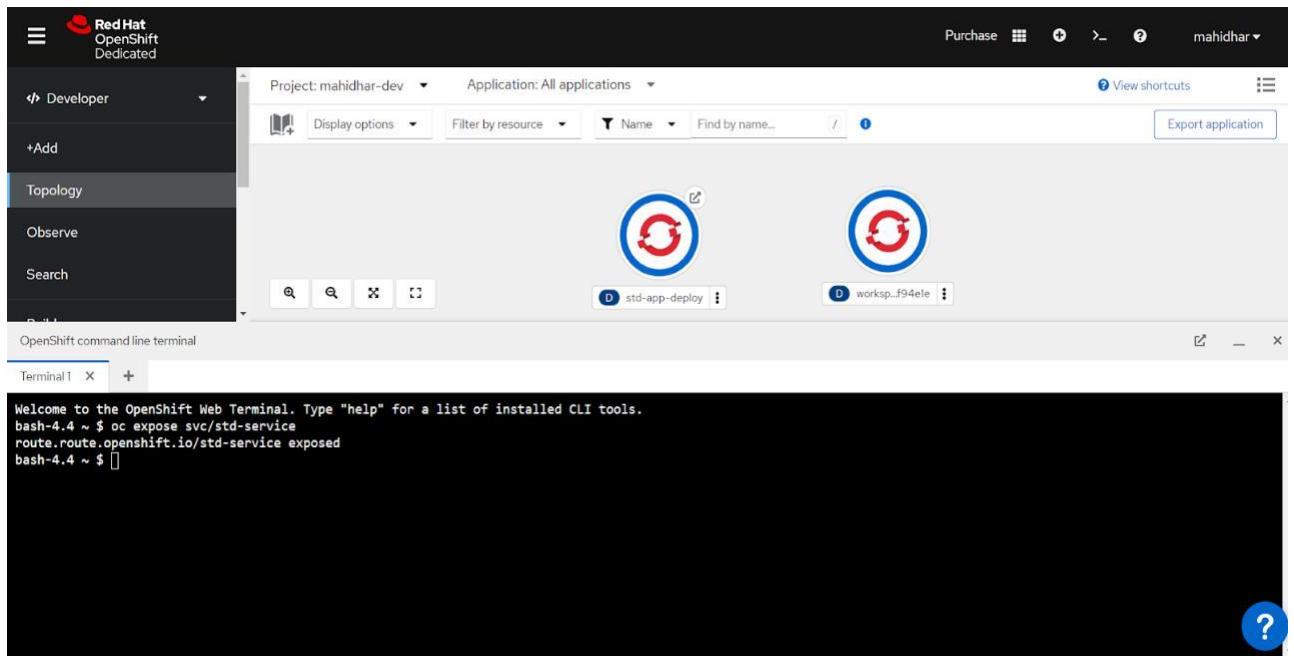
The screenshot shows the Red Hat OpenShift Dedicated web interface with the "Topology" option selected in the sidebar. The main area displays a summary of deployed resources. It includes sections for "Pods" (listing three "std-app-deploy" pods, each running and with a "View logs" link), "Services" (listing one "std-service" pod with port mappings), and "Routes" (indicating no routes are found). The top navigation bar shows "Project: mahidhar-dev" and "Application: All applications". The top right corner has a "View shortcuts" button, a "Purchase" button, and a user icon "mahidhar". A blue question mark icon is in the bottom right corner of the main content area. The "Terminal" button in the navigation bar is highlighted with an orange box.

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.

Red Hat
OpenShift
Dedicated

Project: mahidhar-dev Application: All applications

Developer +Add Topology Observe Search Builds Pipelines Helm Project ConfigMaps Secrets Desktop Tools

Display options Filter by resource Name Find by name... View shortcuts Export application

P std-app-deploy-5bbc5f88c6-rv4zr Running View logs

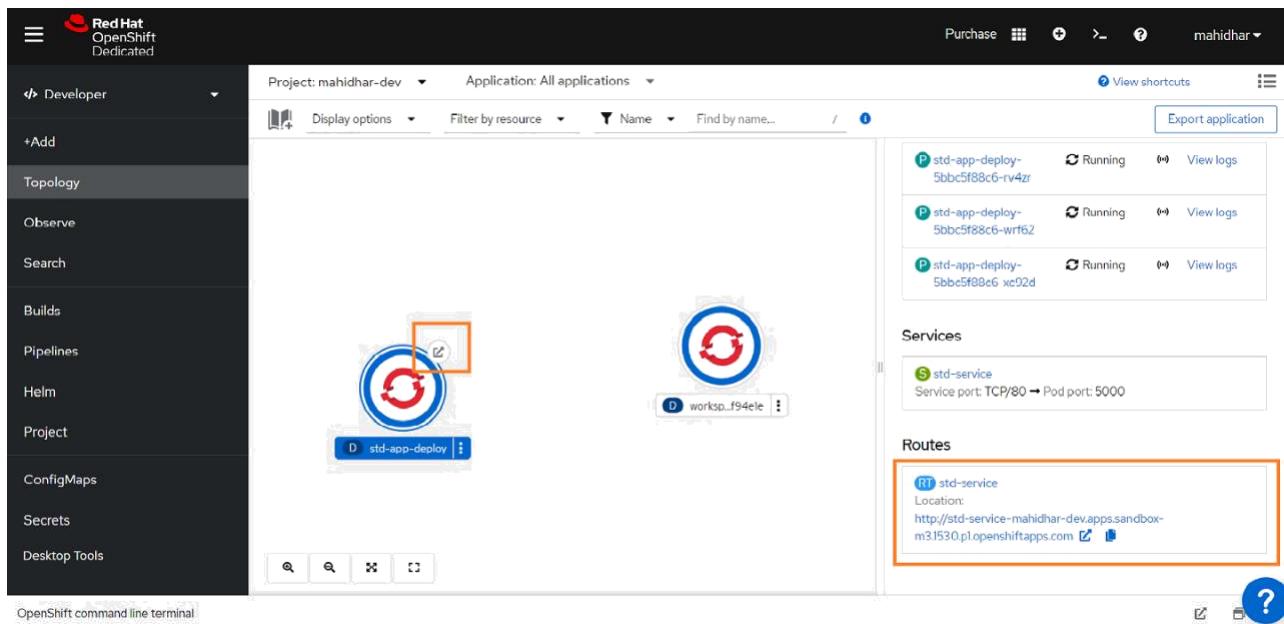
P std-app-deploy-5bbc5f88c6-wrt62 Running View logs

P std-app-deploy-5bbc5f88c6-xc92d Running View logs

S std-service Service port: TCP/80 → Pod port: 5000

R std-service Location: http://std-service-mahidhar-dev.apps.sandbox-m3.1530.p1.openshiftapps.com

OpenShift command line terminal



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

