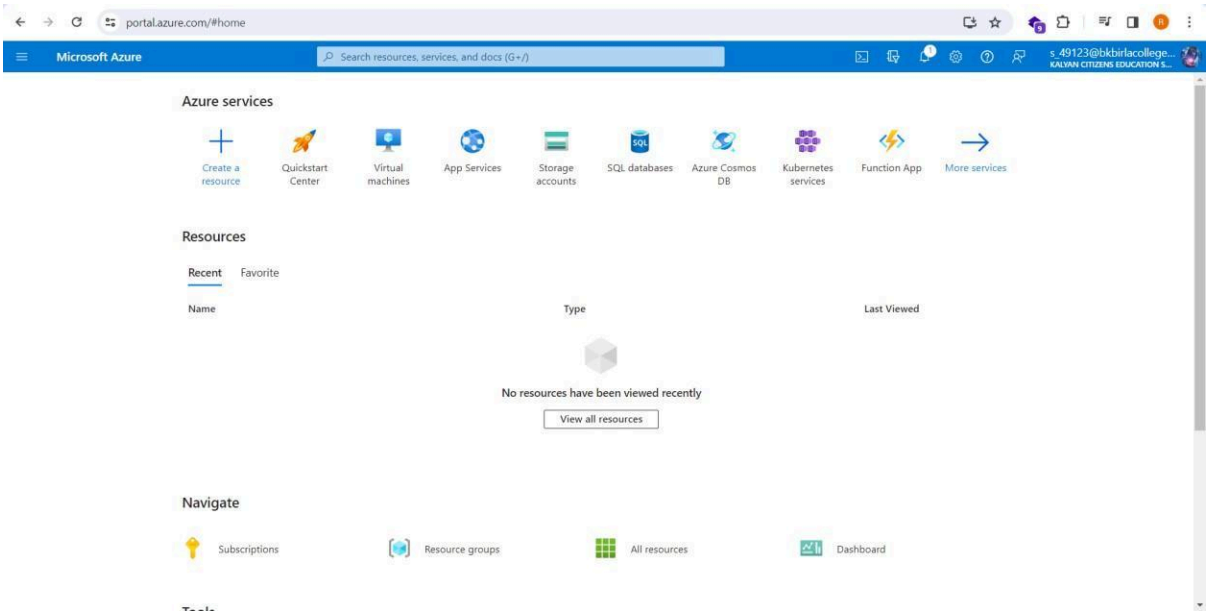
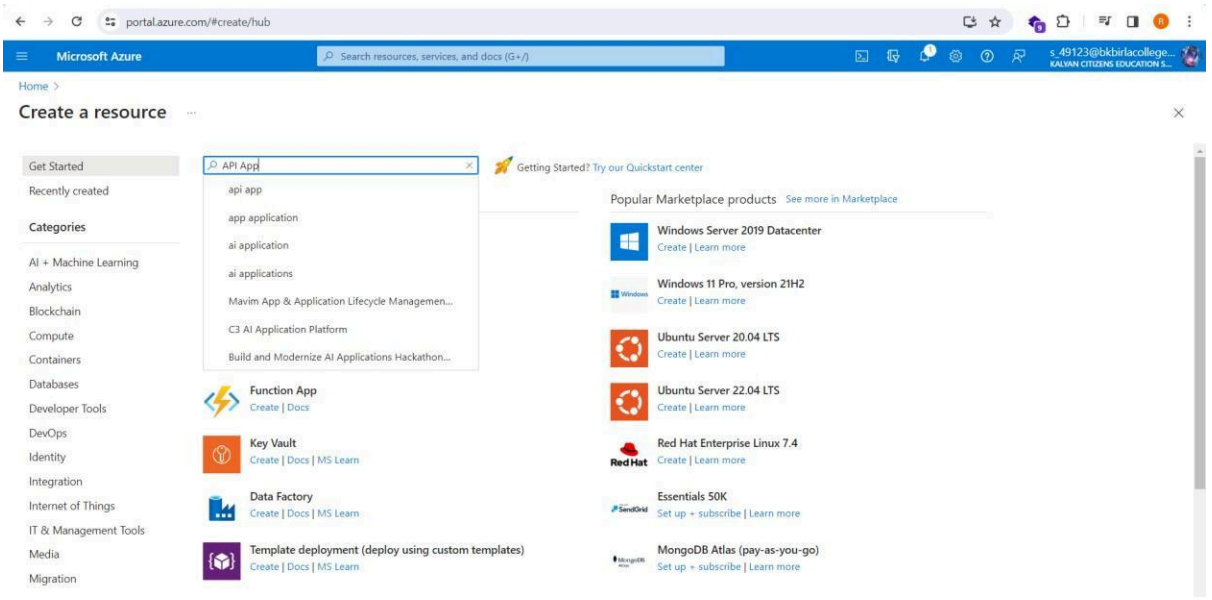
Practical 1
Create a simple Azure API Application

Theory: Azure API Applications provide a streamlined approach to building and deploying APIs on the Azure cloud platform. These applications leverage
Azure's robust infrastructure to handle API requests and responses efficiently. With Azure API Applications, developers can easily create RESTful APIs using popular frameworks like ASP.NET Core and Node.js, while also taking advantage of Azure's scalable and reliable architecture. This enables developers to focus more on building the core functionality of their APIs without worrying about
infrastructure management. Additionally, Azure API Applications offer features such as automatic scaling, authentication, and monitoring, ensuring that APIs remain performant and secure. Overall, Azure API Applications simplify the process of creating and managing APIs, empowering developers to deliver high- quality services to their users.

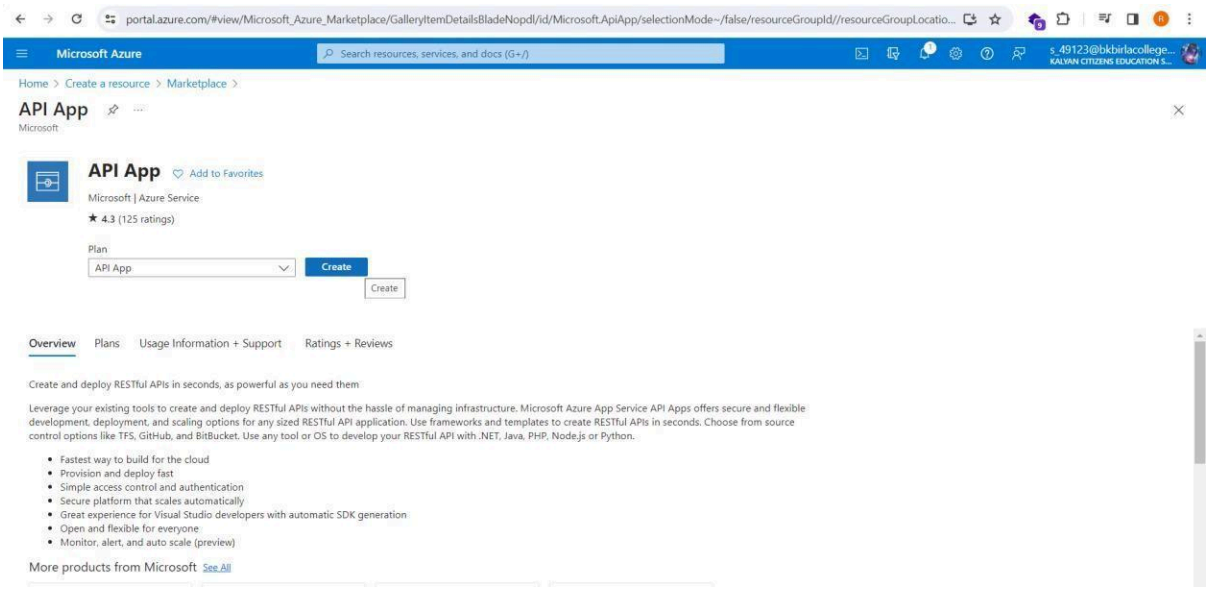
Step1: Go to Azure portal i.e. https://www.portal.azure.com/ and login with your Azure Account.



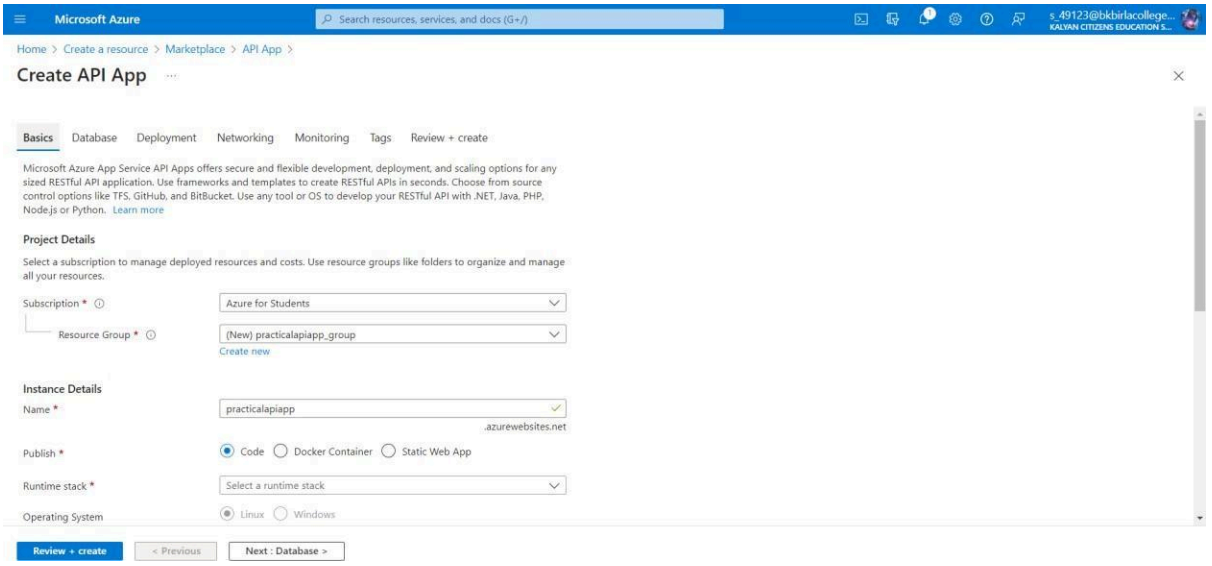
Step2: Now go to “Create a resource” and search for “API App”.



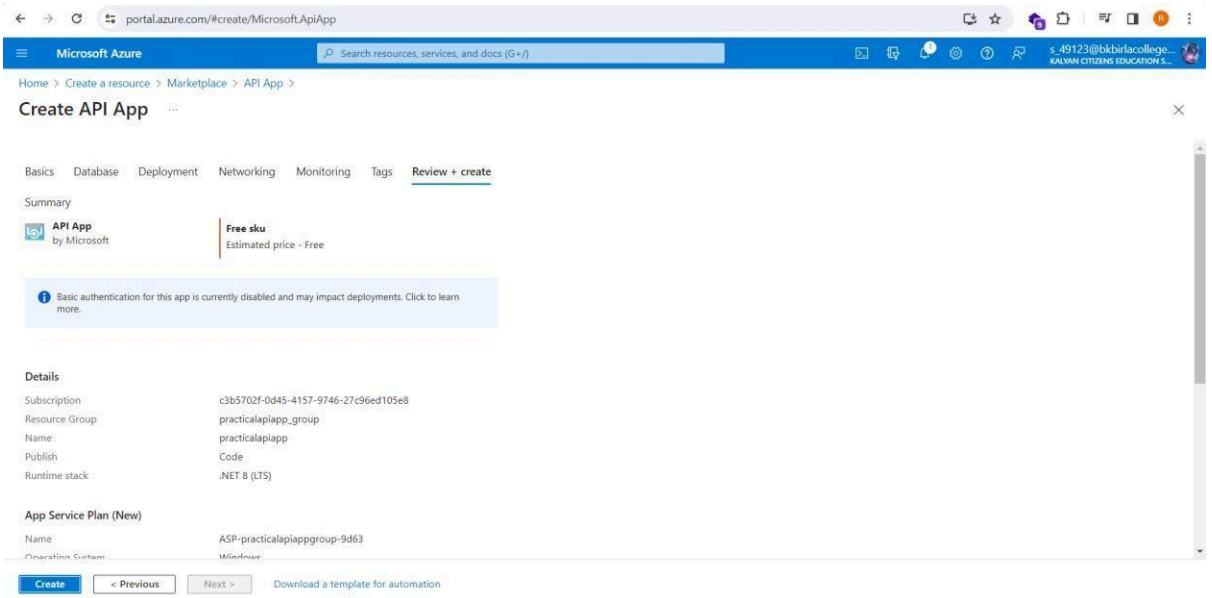
Step3: Now click on create and create a new API App.



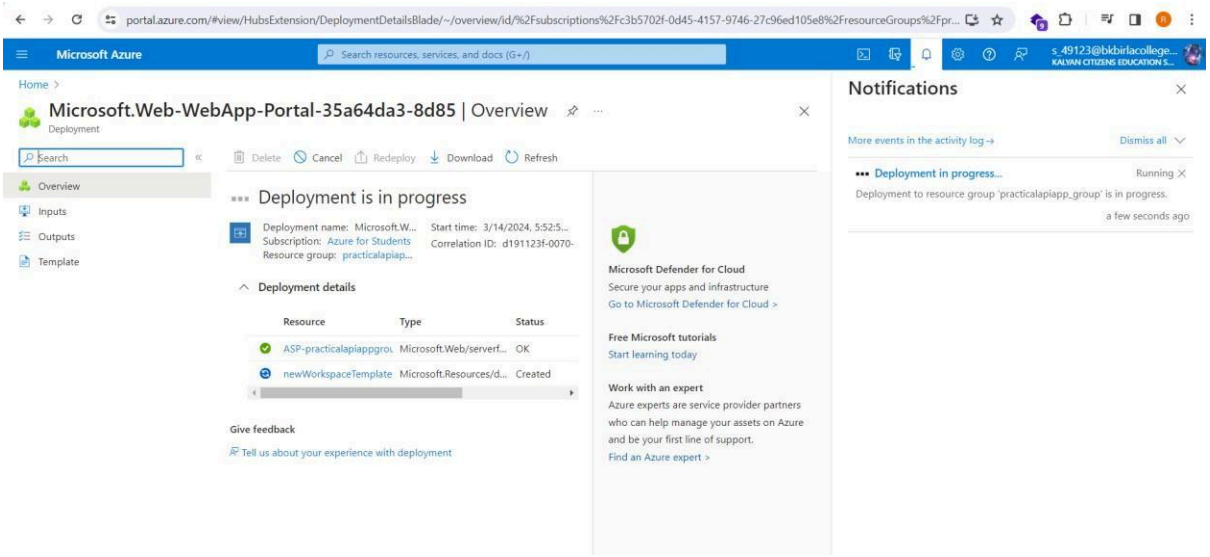
Step4: Now give your API app a name and click on “review + create”.



Step5: Now click on create.

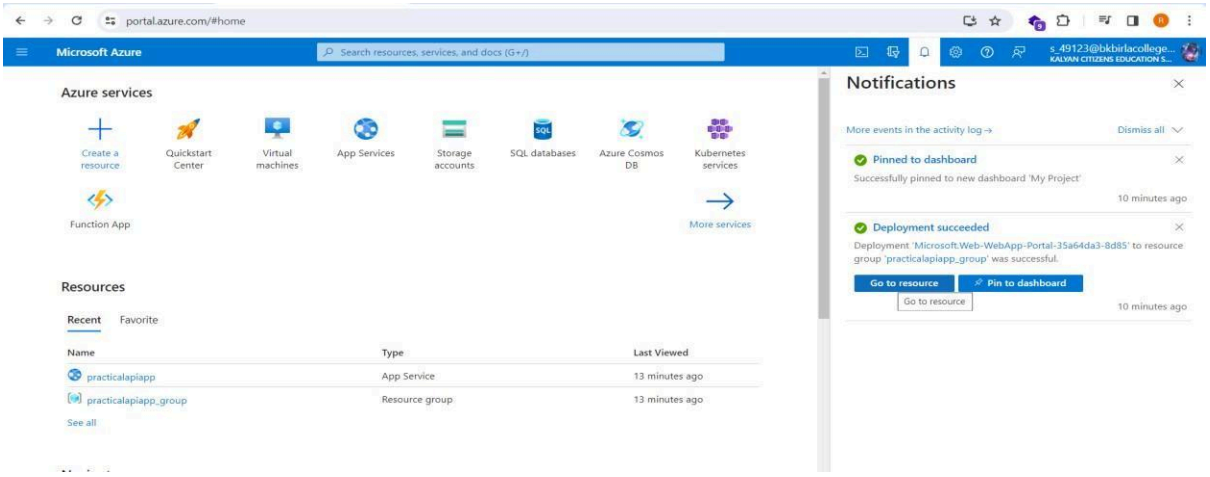


Step6: Now you can see your app deployment is in progress.

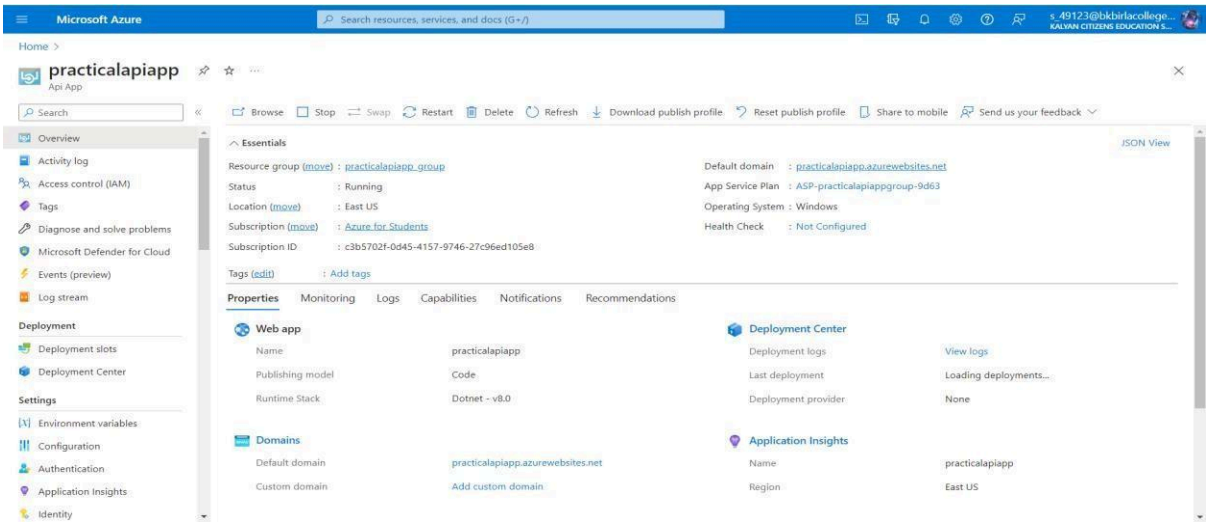


Wait till the deployment is completed.

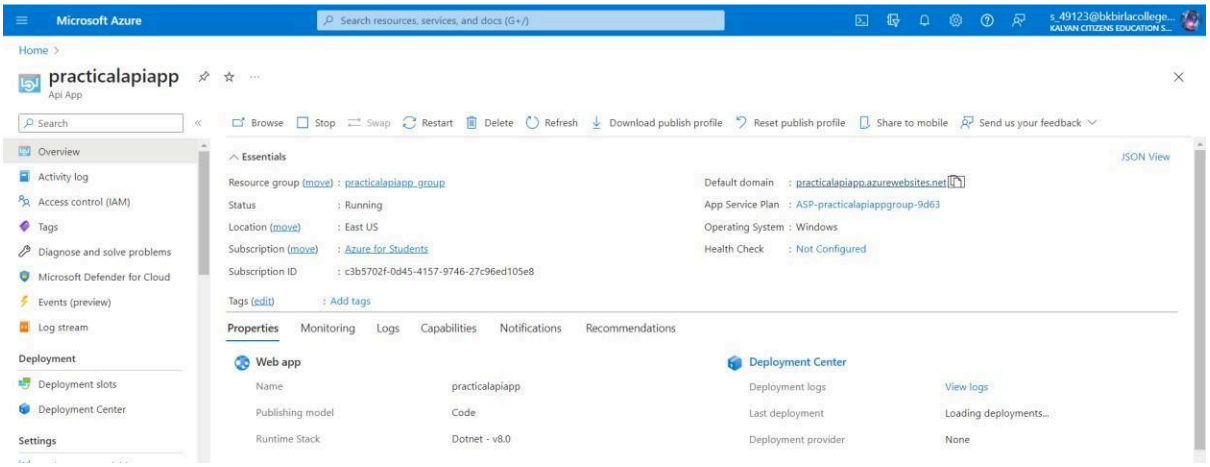
Step7: Once the deployment is completed click on “Go to resources”.

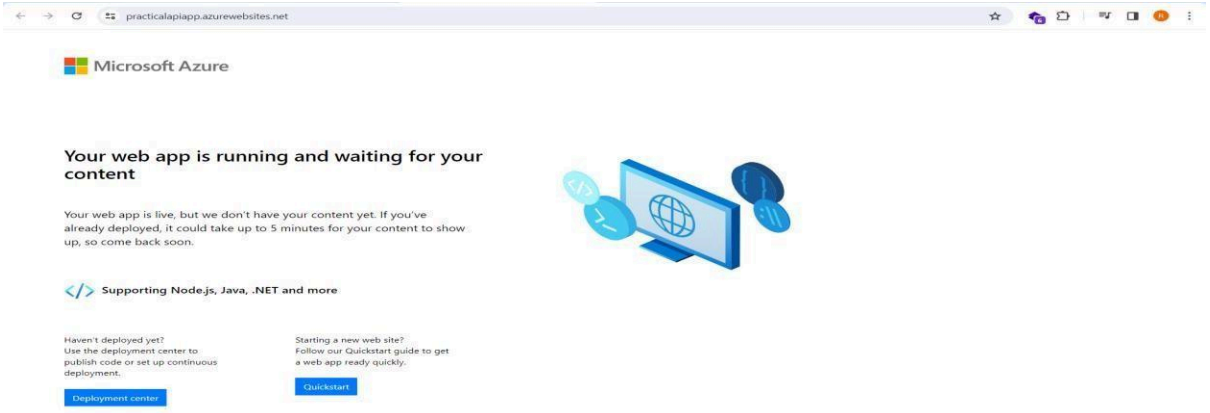


Now you will see a window like this:

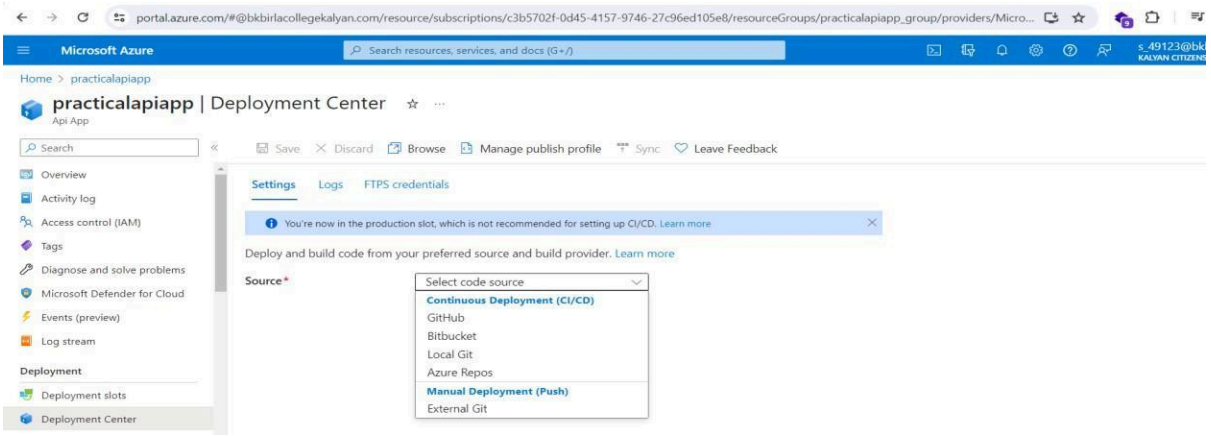


Step8: Now you can click on your default domain and it will take you to your web applications default webpage.

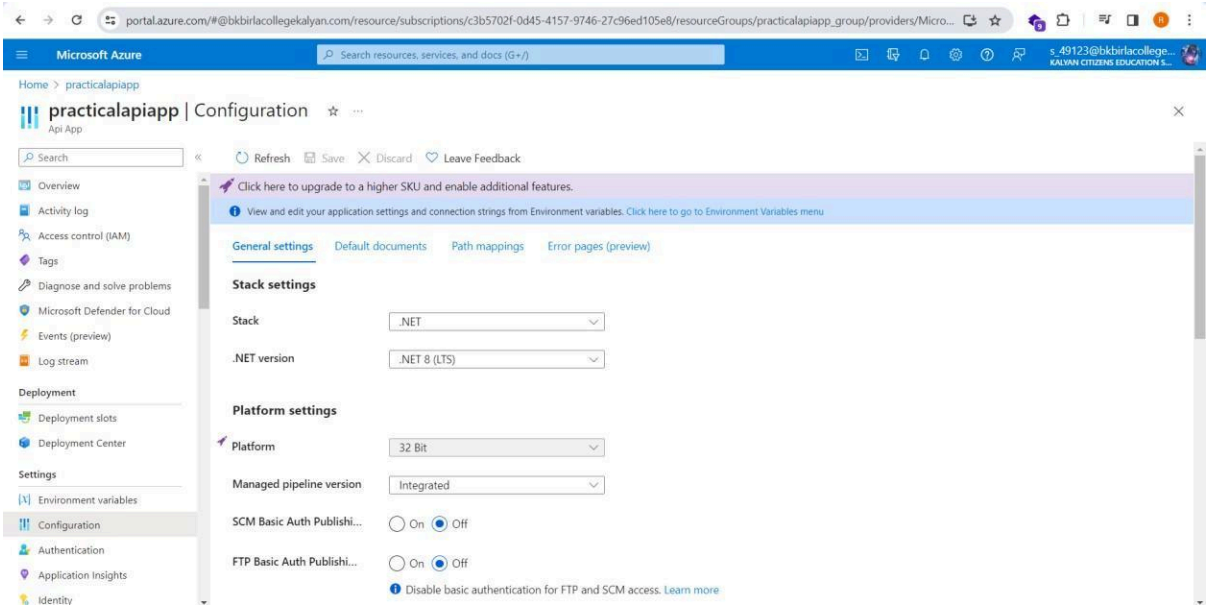




Step9: Now click on deployment center and now on this tab you can add your code source via GitHub or from any other platform.



Step10: Now click on Configuration and here you can configure your application.

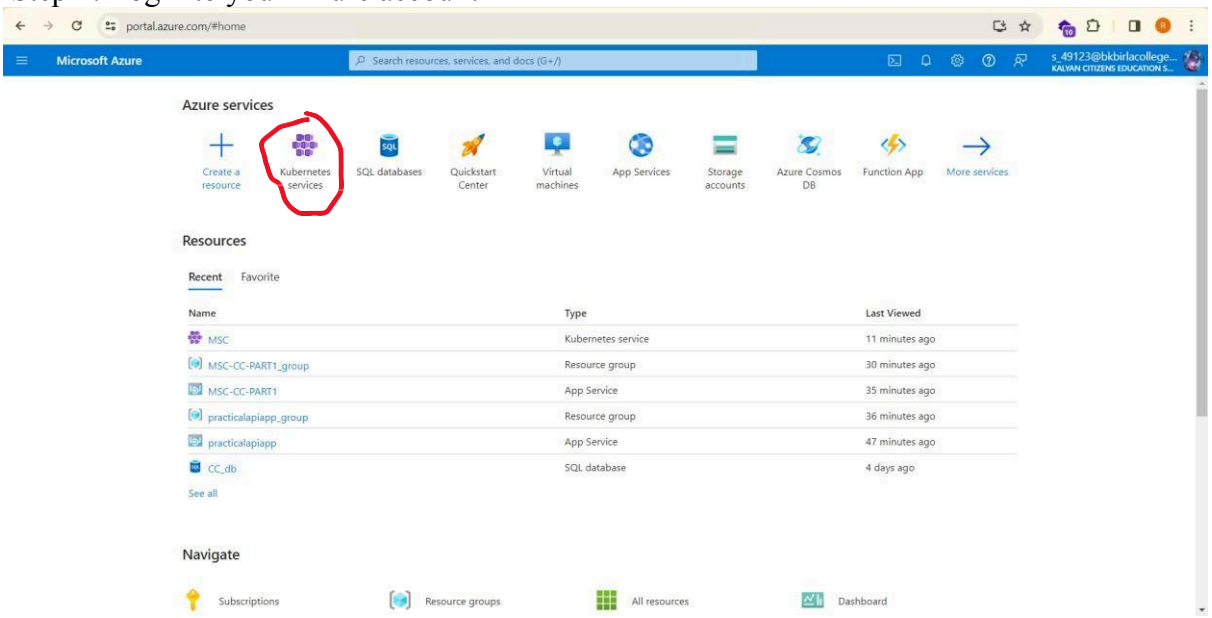


Practical 2

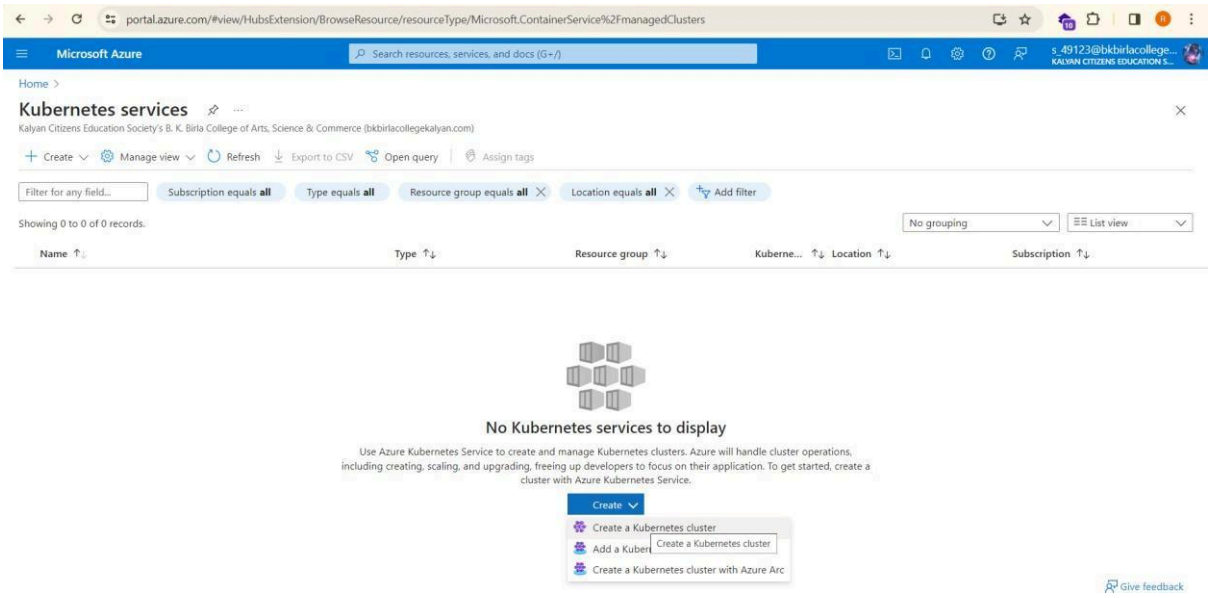
Create an Azure Kubernetes Cluster and integrate it with our Azure API App.

Theory: An Azure Kubernetes Cluster (AKS) is a managed
container orchestration service provided by Microsoft Azure, allowing users to deploy, manage, and scale containerized applications using Kubernetes. With AKS, users can abstract away the complexities of managing Kubernetes infrastructure and focus on deploying and running their applications efficiently. It automates tasks such as cluster setup, scaling, and maintenance, providing built-in monitoring, logging, and security features. AKS integrates seamlessly with other Azure services,
enabling users to leverage functionalities such as Azure Active Directory for authentication, Azure Monitor for monitoring, and Azure DevOps for continuous integration and deployment pipelines. This managed service empowers developers and operators to streamline the deployment and management of
containerized applications, fostering agility, scalability, and reliability in cloud-native development workflows.

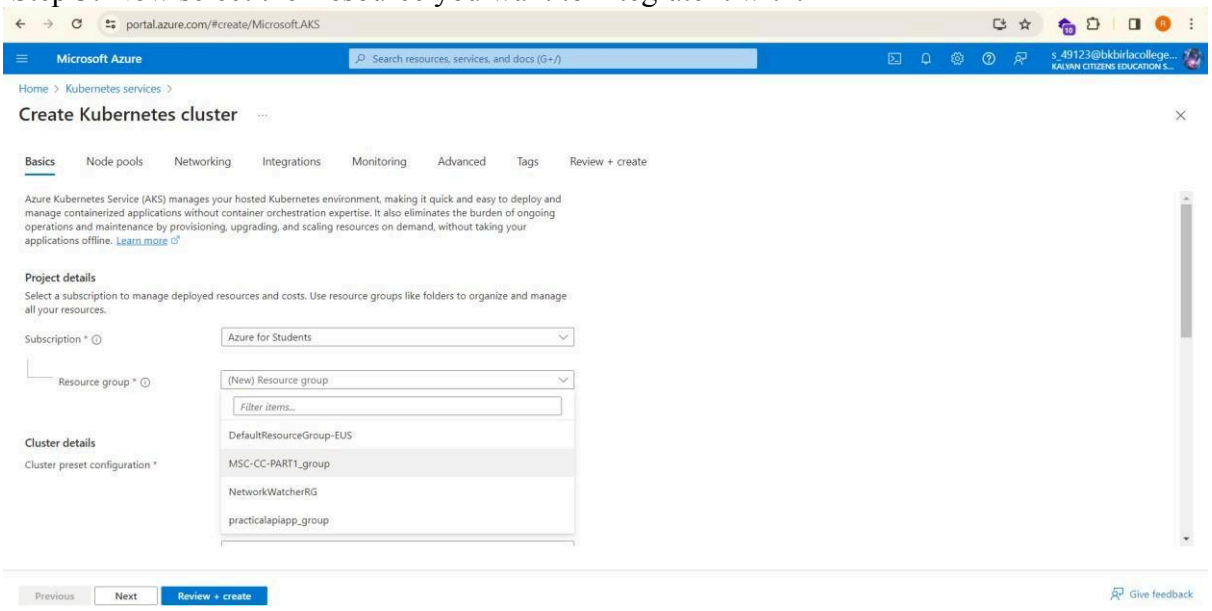
Step 1: Login to your Azure account



Step 2: Click on Kubernetes Services and then click on “Create a Kubernetes Cluster”

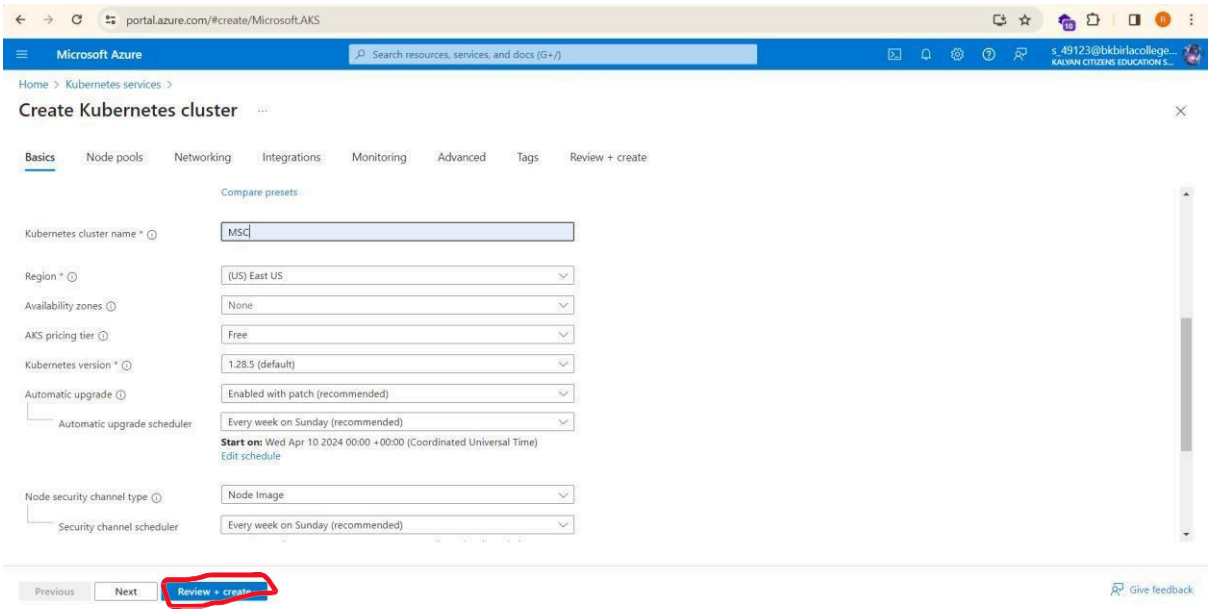


Step 3: Now select the Resource you want to integrate it with.

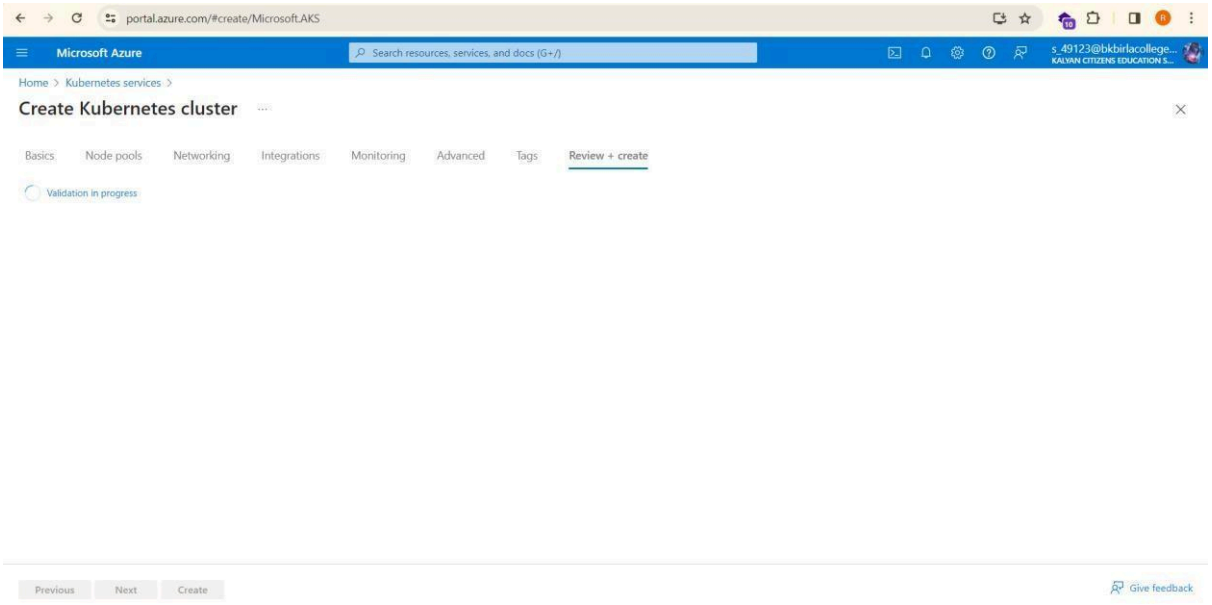


For e.g. here we have selected MSC-CC-PART1_group

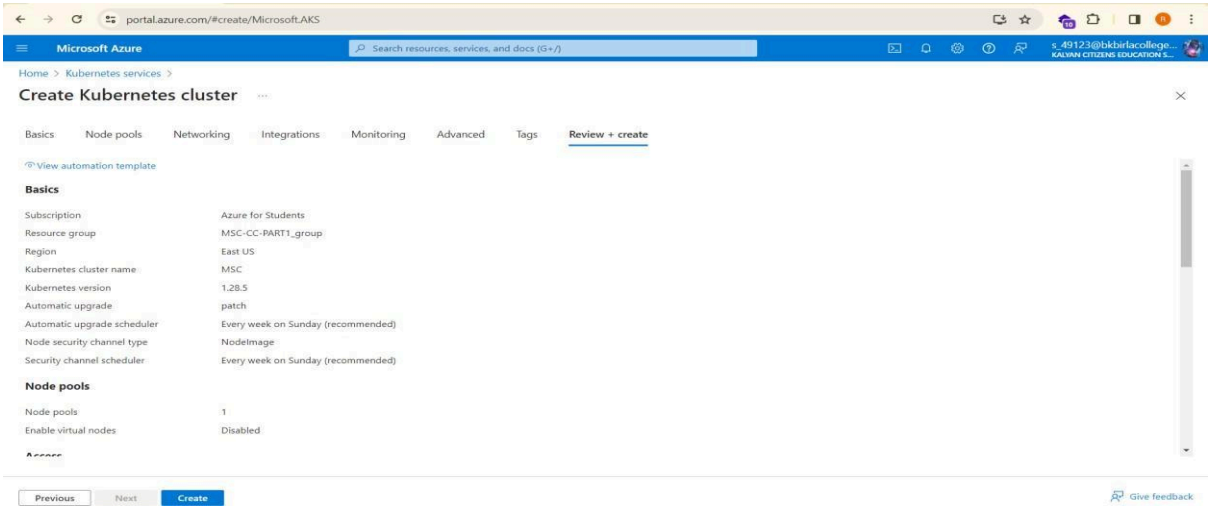
Step 4: Now give you Kubernetes Cluster a name and Click on “Review + Create”



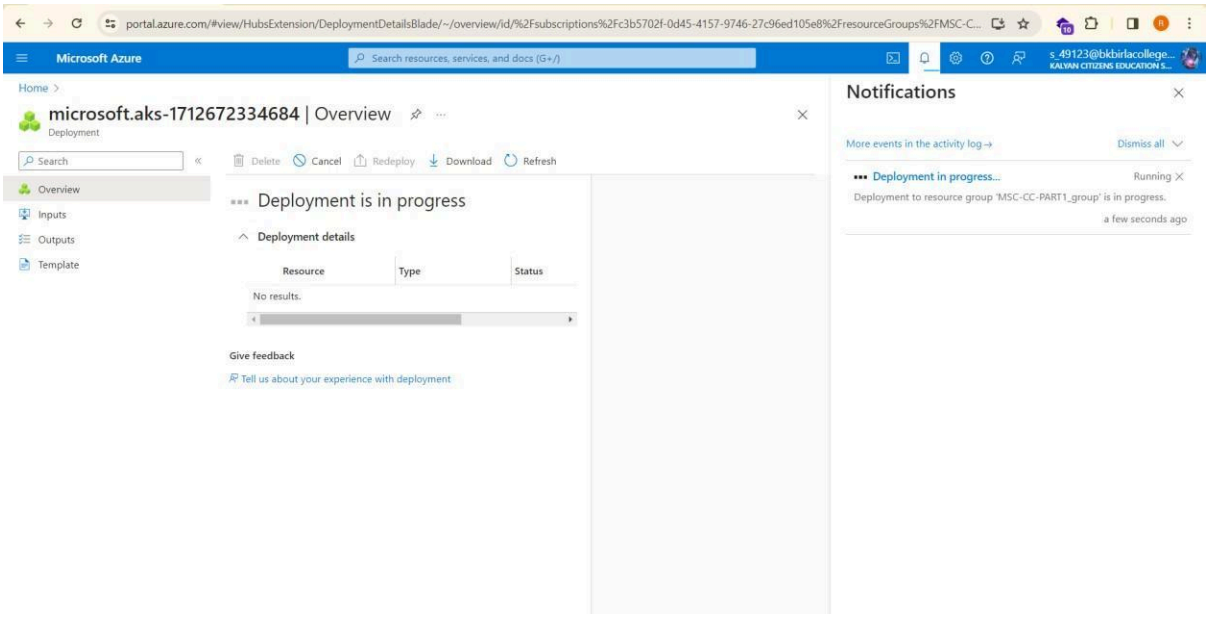
For e.g. here we have given it the name MSC.

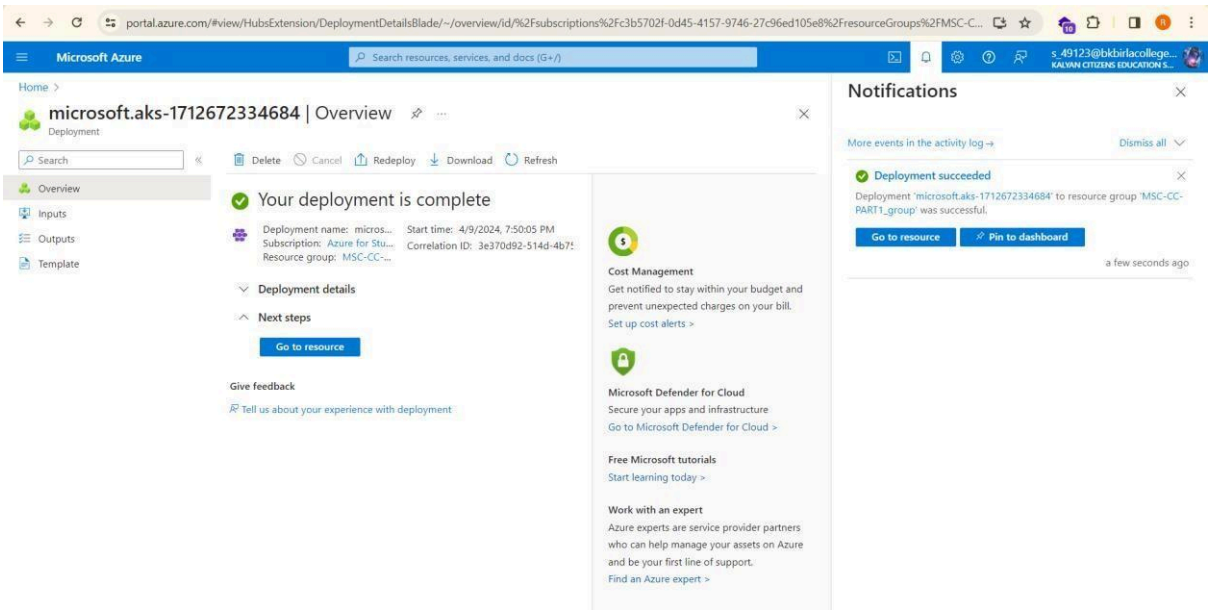


Step 5: Now click on Create.

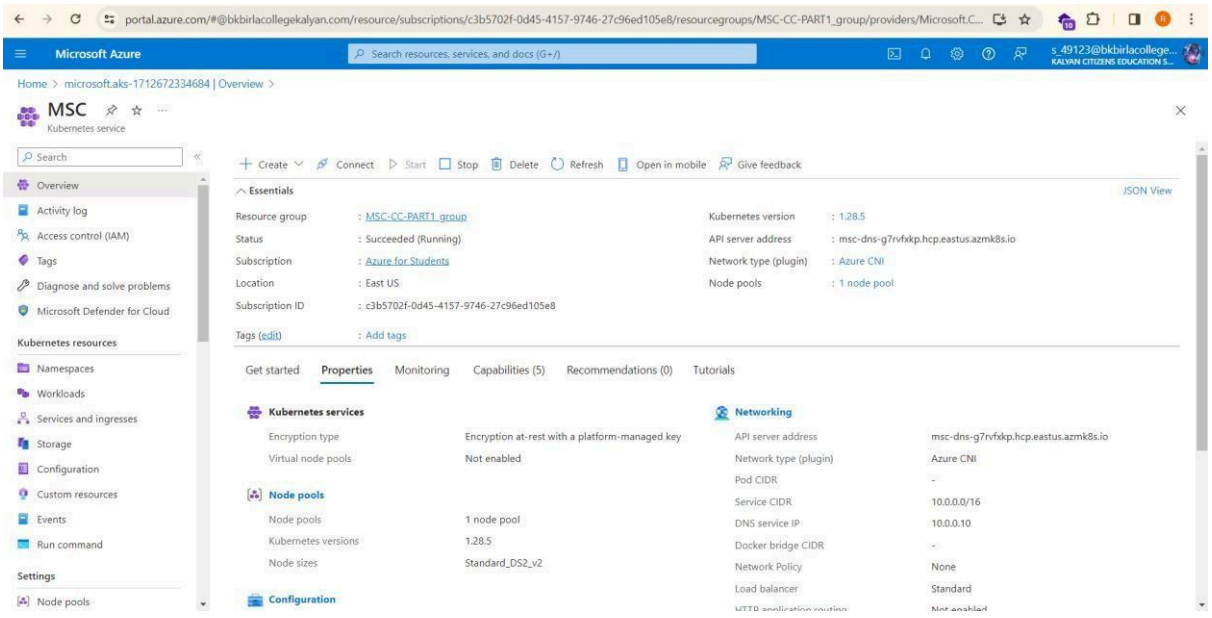


Step 6: Now wait till the deployment finishes.





Step 7: Now you can see that our Azure API App is successfully connected with our Azure Kubernetes Clusters.

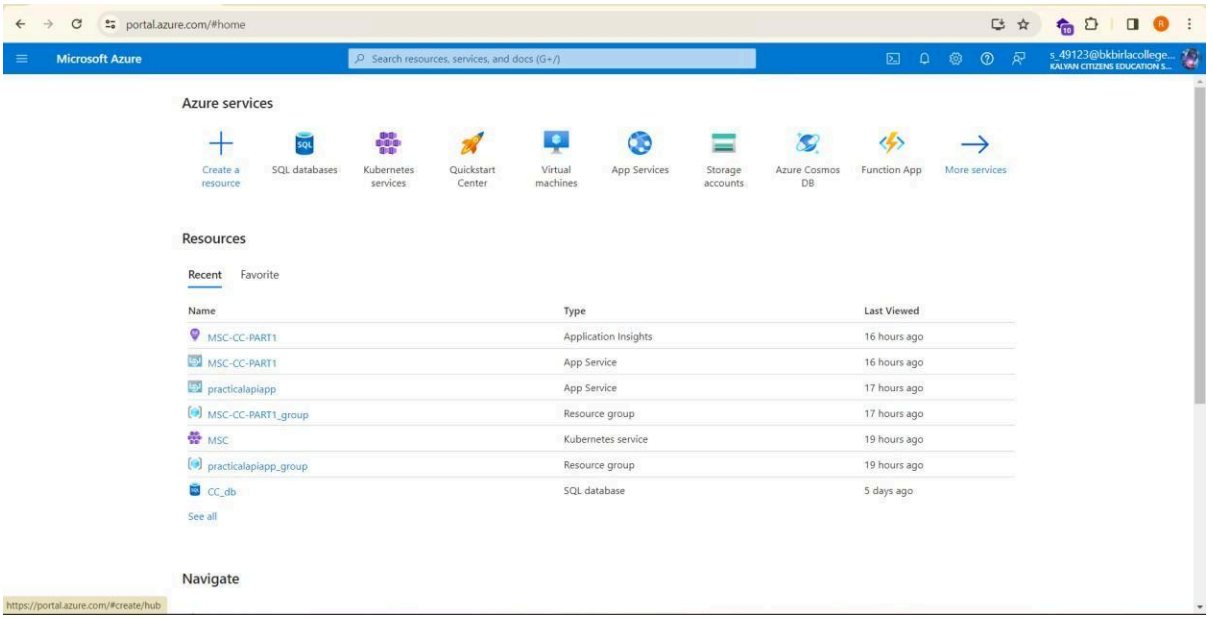


Practical 3

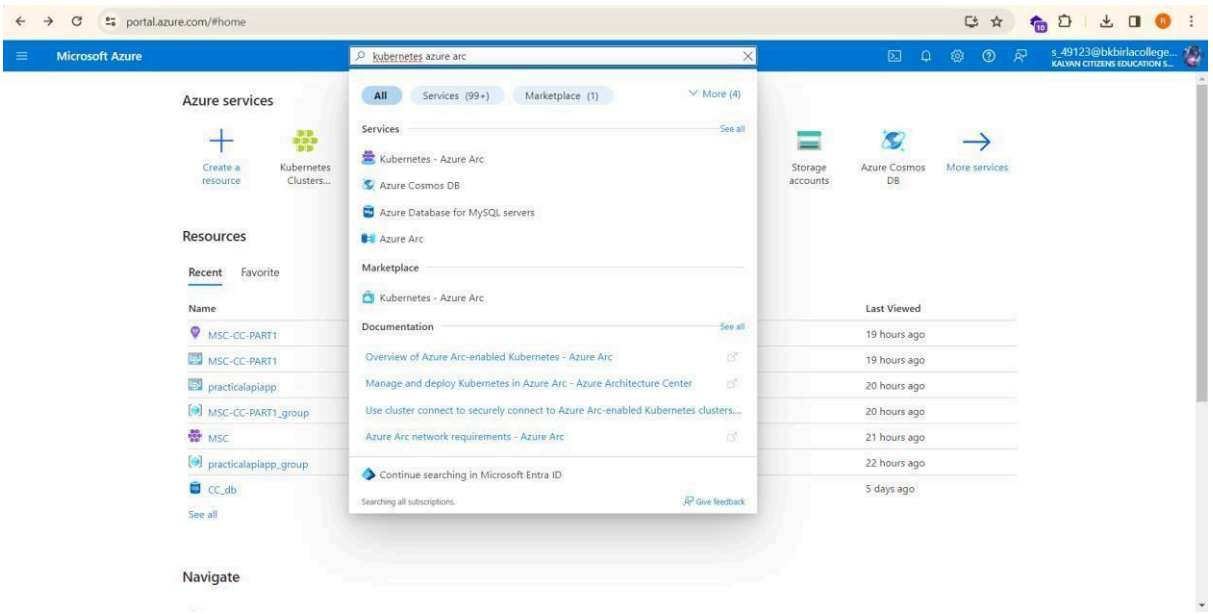
Create and add an Azure Kubernetes Cluster with Azure Arc

Theory: To create and add an Azure Kubernetes Cluster (AKS) with Azure Arc, you would first create an AKS cluster using the Azure portal, Azure CLI, or Azure Resource Manager templates. Then, you would enable Azure Arc for Kubernetes on the AKS cluster by navigating to the Azure Arc-enabled Kubernetes service in the Azure portal and selecting "Add Azure Kubernetes Cluster." Follow the prompts to specify the subscription, resource group, and other details, and Azure Arc will automatically onboard the AKS cluster, allowing you to manage and monitor it alongside your other Arc-enabled resources from a centralized Azure management interface.

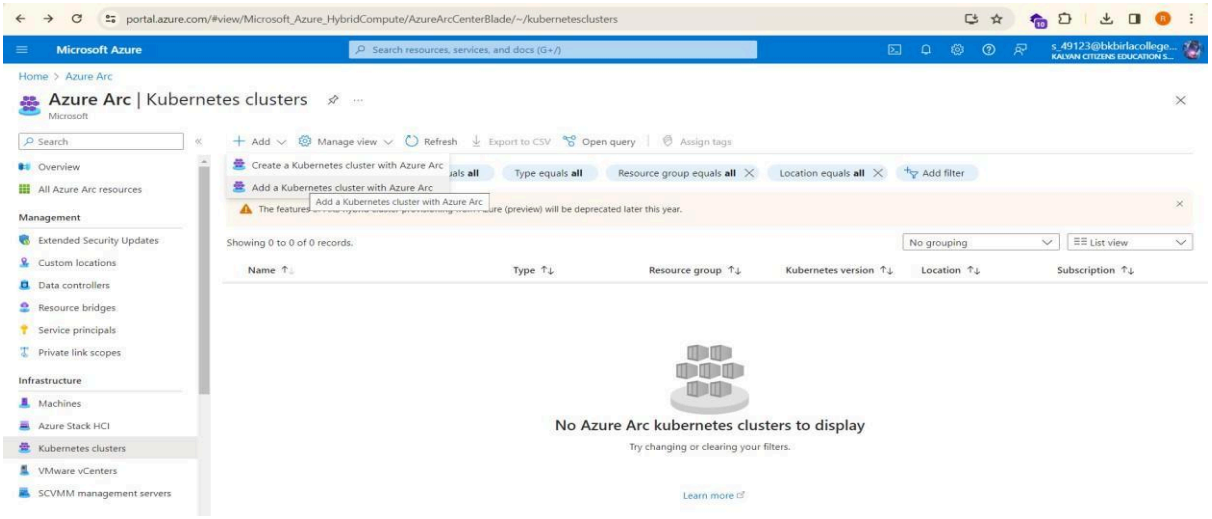
Step 1: Login to your Azure account on Azure Portal.



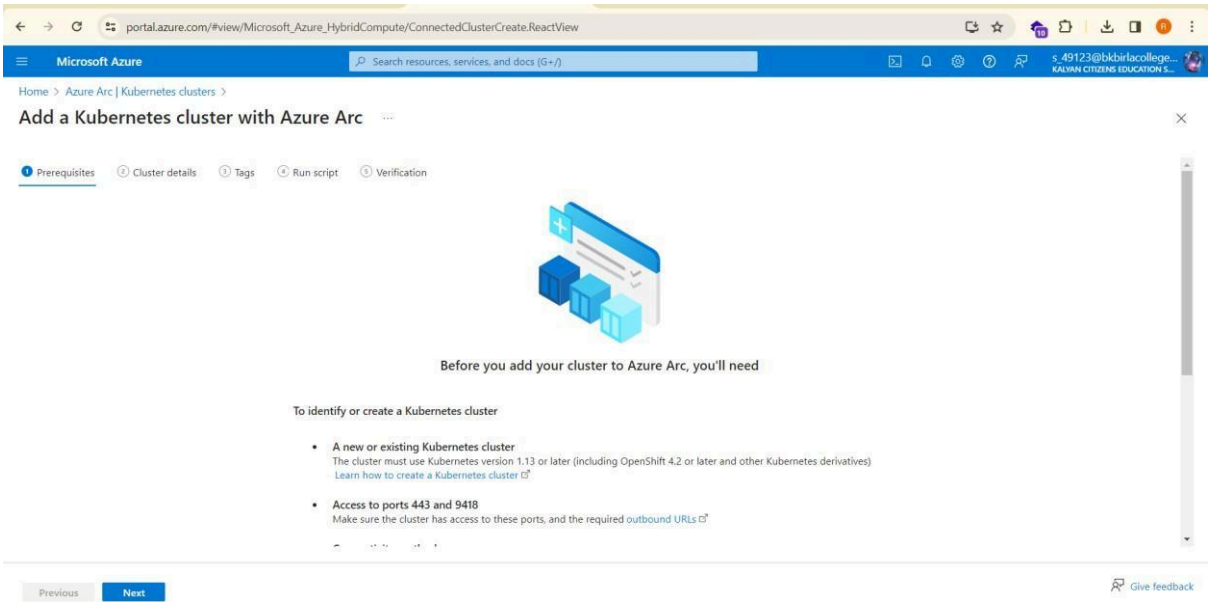
Step 2: click on search and search for Kubernetes Azure Arc and click on it.



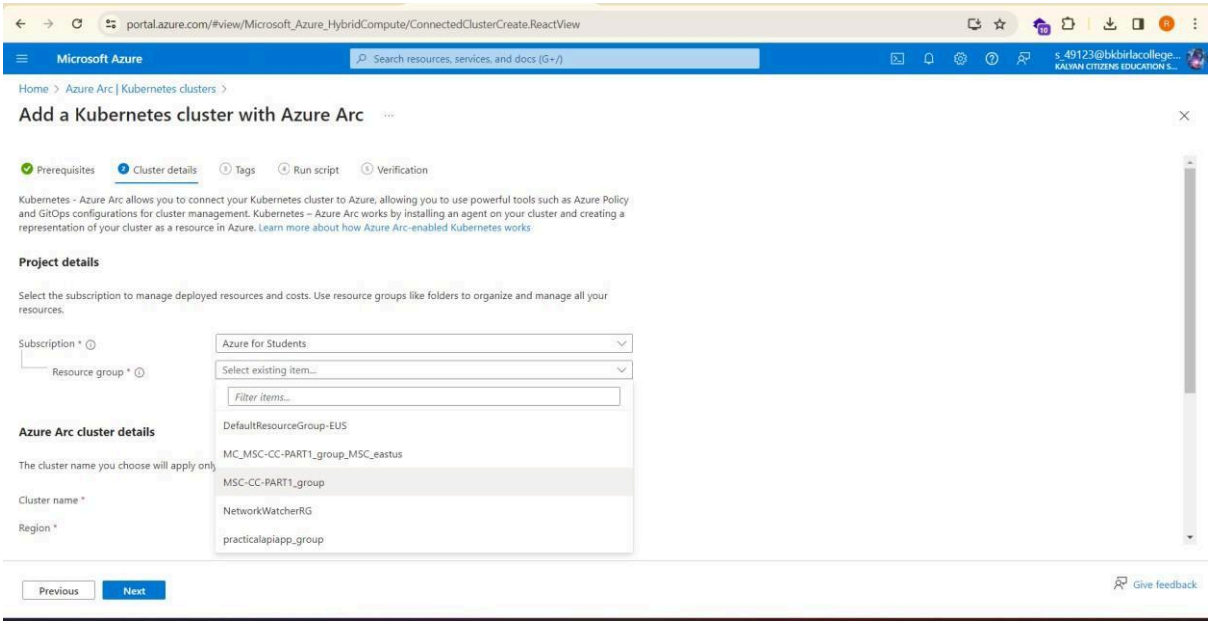
Step 3: Now click on Add a “Kubernetes Cluster with Azure Arc”



Step 4: Now click on “next.”

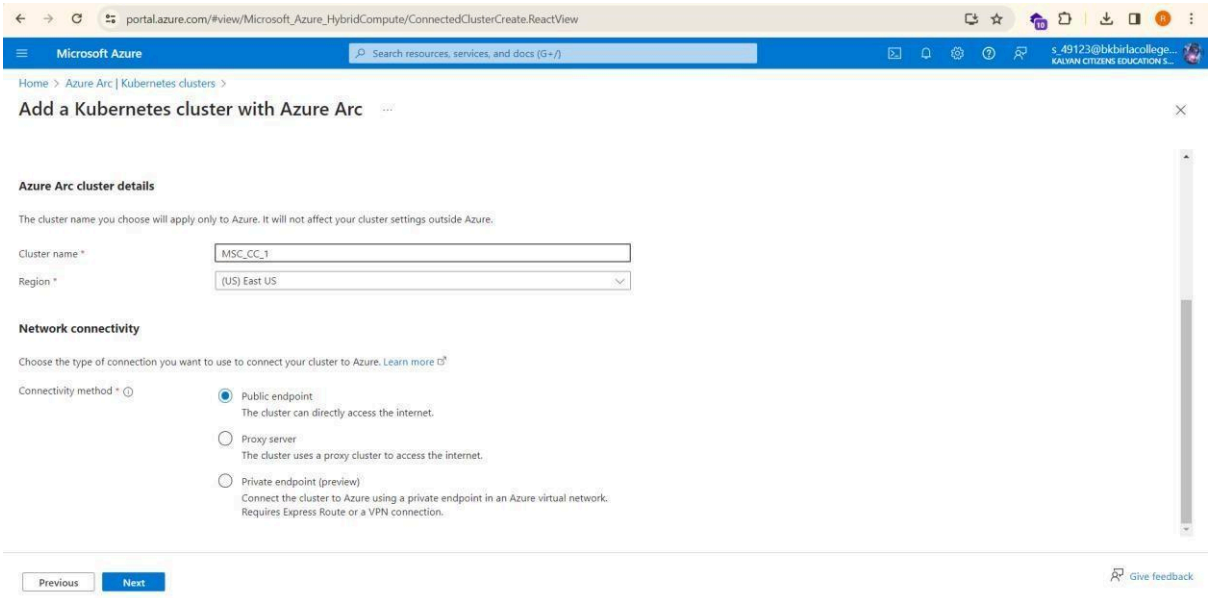


Step 5: Now select a Resource group for your Kubernetes cluster.



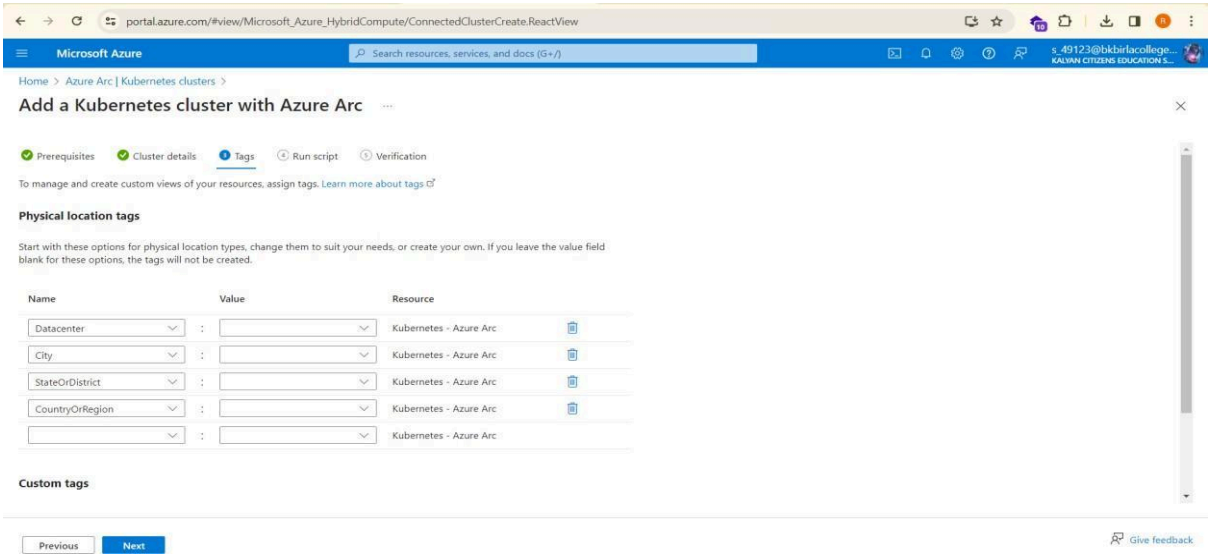
For e.g. here we have selected MSC-CC-PART1_group

Step 6: Now give your Kubernetes Cluster a name and click on next.

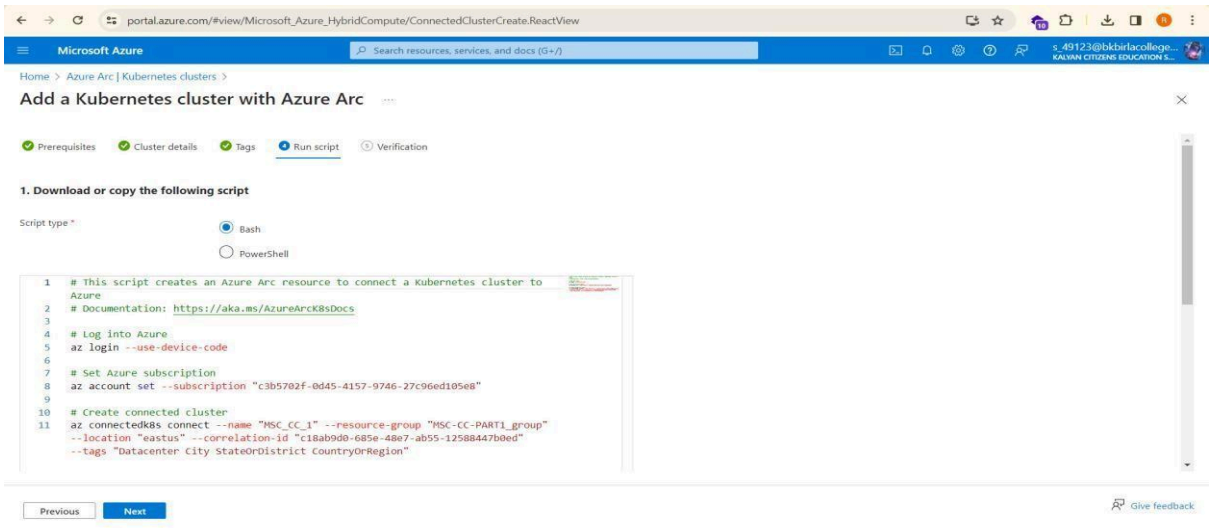


For e.g. here we have given the name “MSC_CC_1”

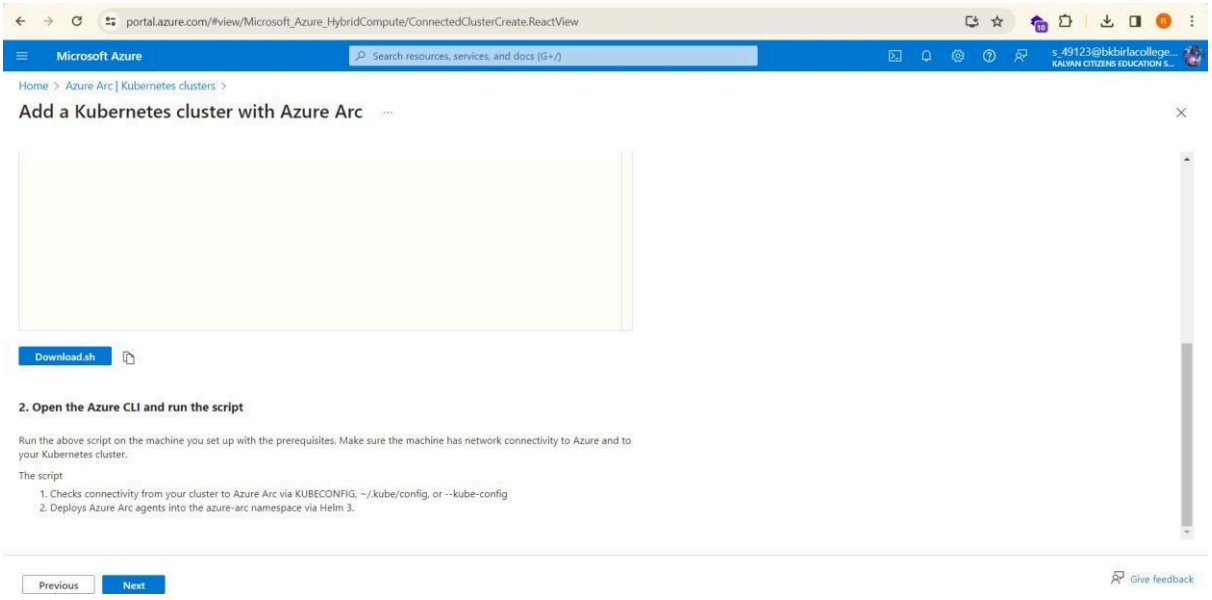
Step 7: Now here you can add physical location tag, or you can directly move to the next step by clicking on next.



Step 8: Here you can see that the bash script is generated successfully.

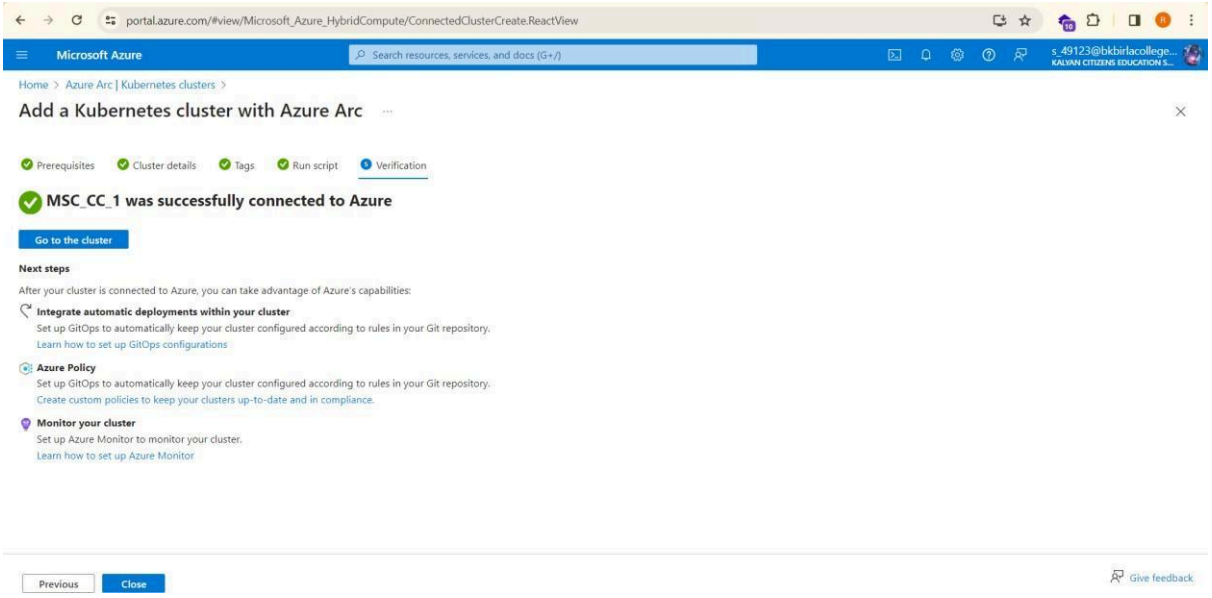


Here you can even download and deploy Azure Arc agents into the azure-arc namespace.

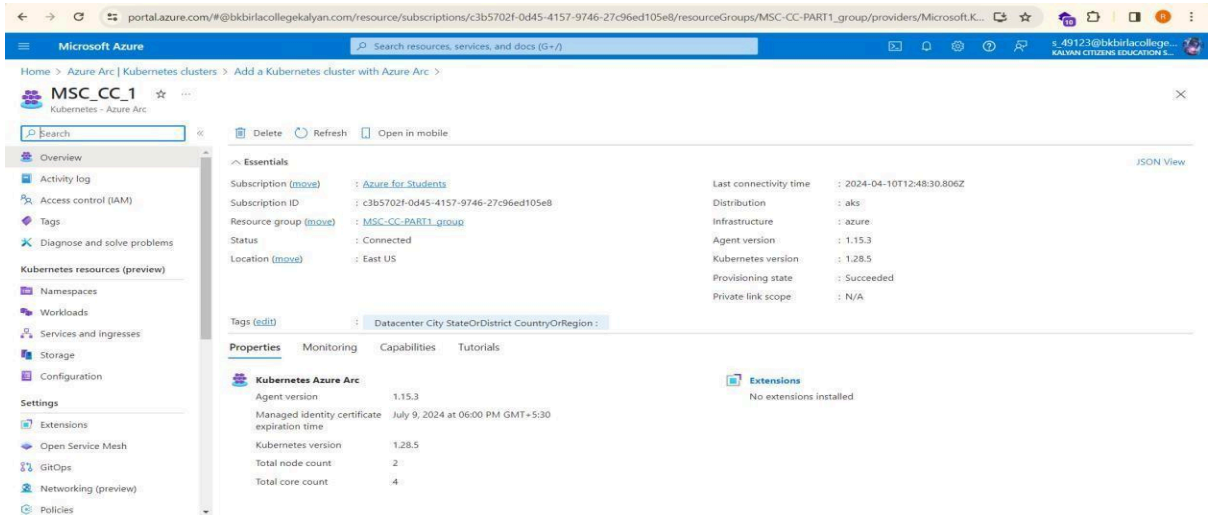


And now click on Next

Step 9: Now you can see that Kubernetes Cluster is successfully connected to Azure.



Now click on cluster and you will be able to see your Azure Kubernetes Cluster.

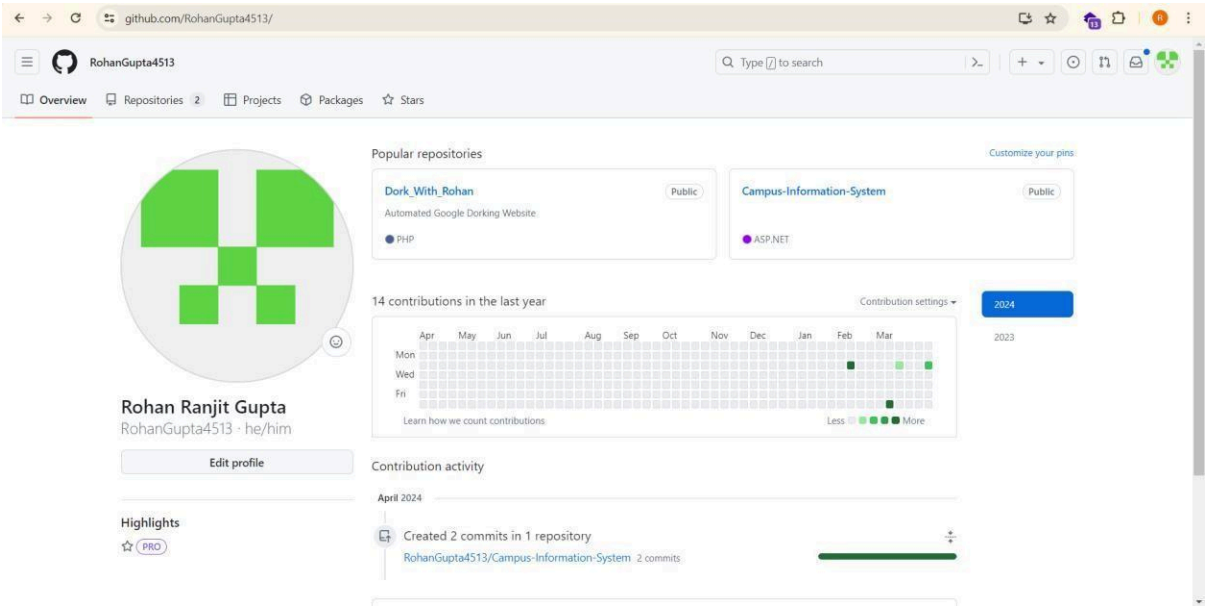


Practical 4

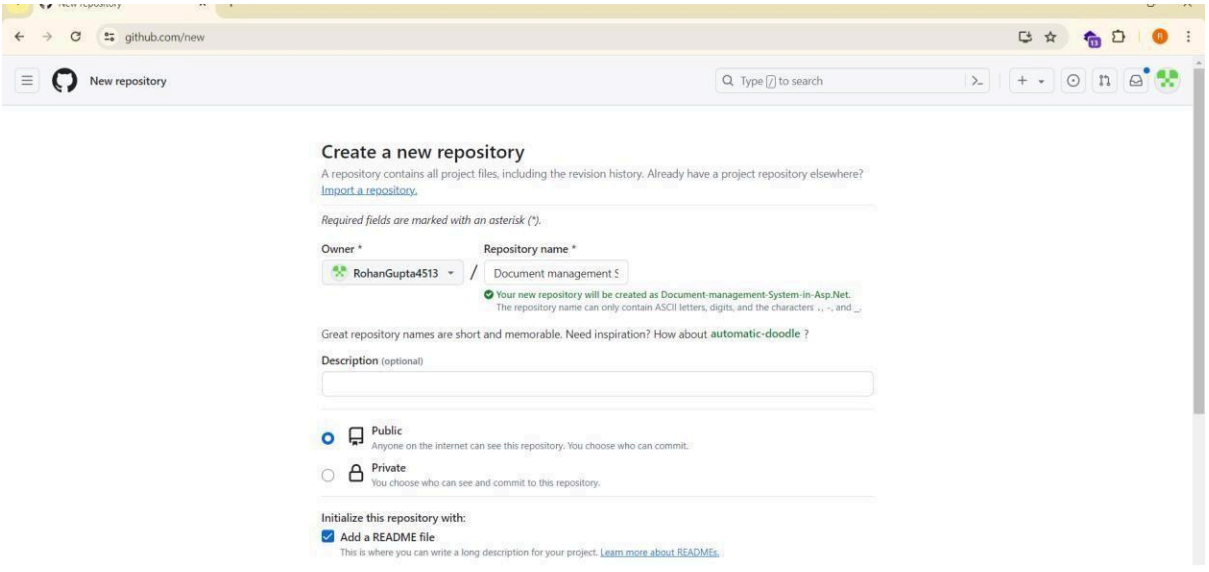
Create a GitHub Repository for our .NET DevOps Web Application

Theory: To create a GitHub repository for your .NET DevOps web application, navigate to GitHub and sign in. Click on the "+" icon in the top- right corner and select "New repository." Provide a name, description, and choose visibility settings for your repository. Optionally, initialize the repository with a README file. If your application already exists locally, follow the instructions to push an existing repository from the command line. Otherwise, clone the repository to your local machine, add your .NET project files, commit the changes, and push them to GitHub. This repository will serve as a centralized location for collaboration, version control, and automated DevOps workflows for your .NET web application.

Step 1: Login to your GitHub account.

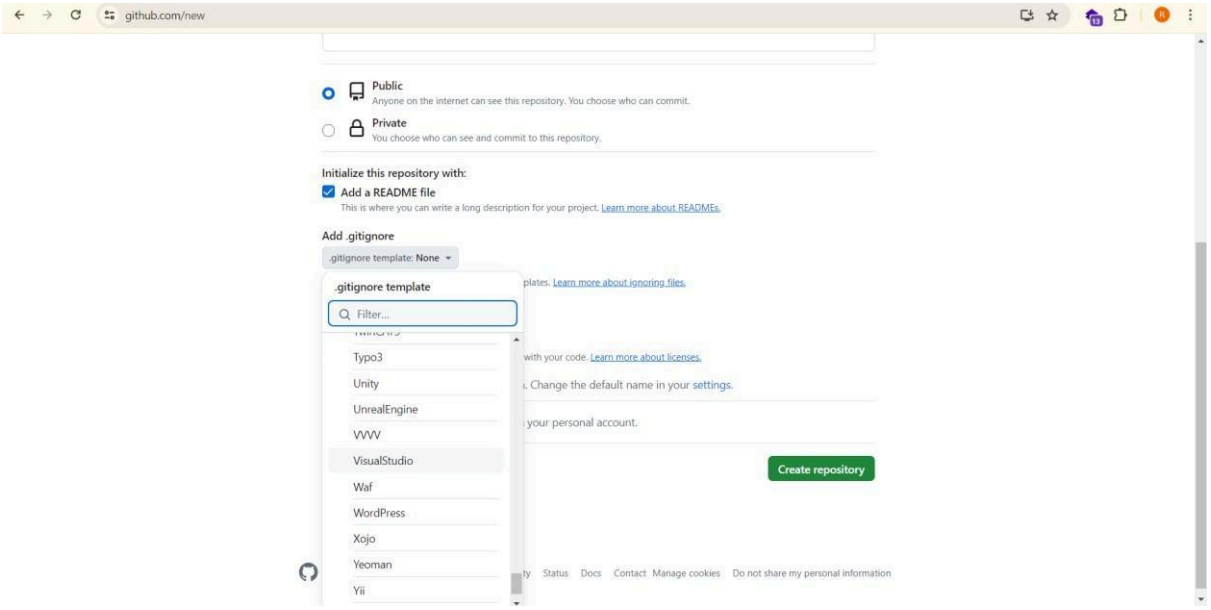


Step 2: Create a new repository.



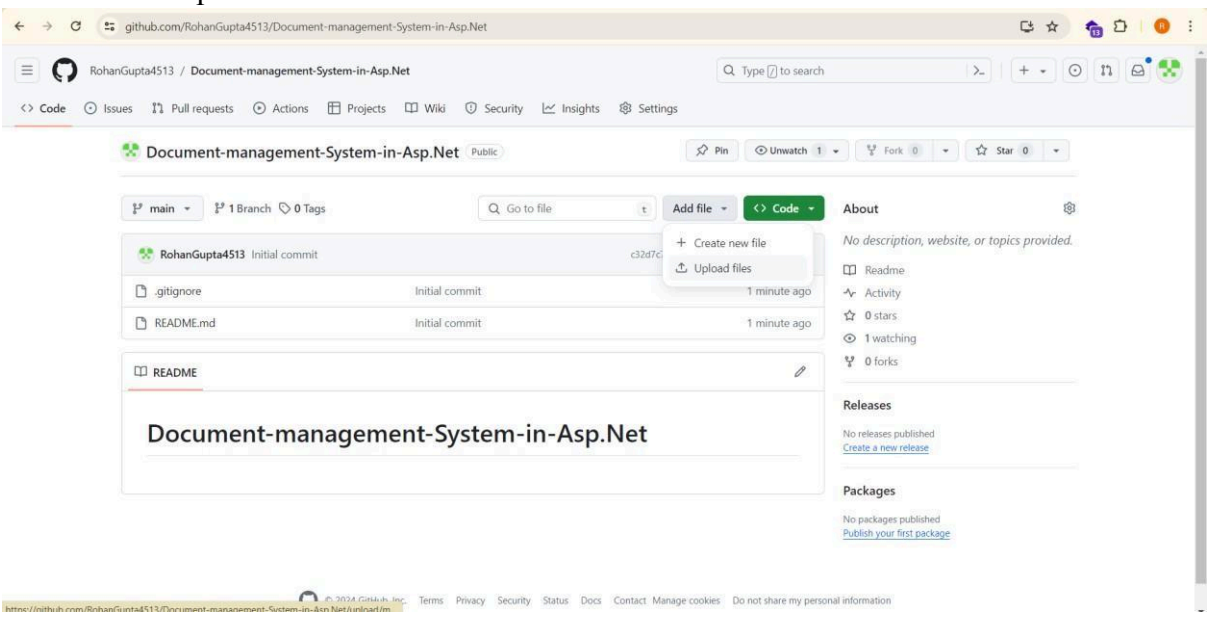
And give your repository a name. Here we have given the name
"_Project_E-Blogging-asp-net-project"

Step 3: Now add a README file and add ". gitignore" and select "VisualStudio"

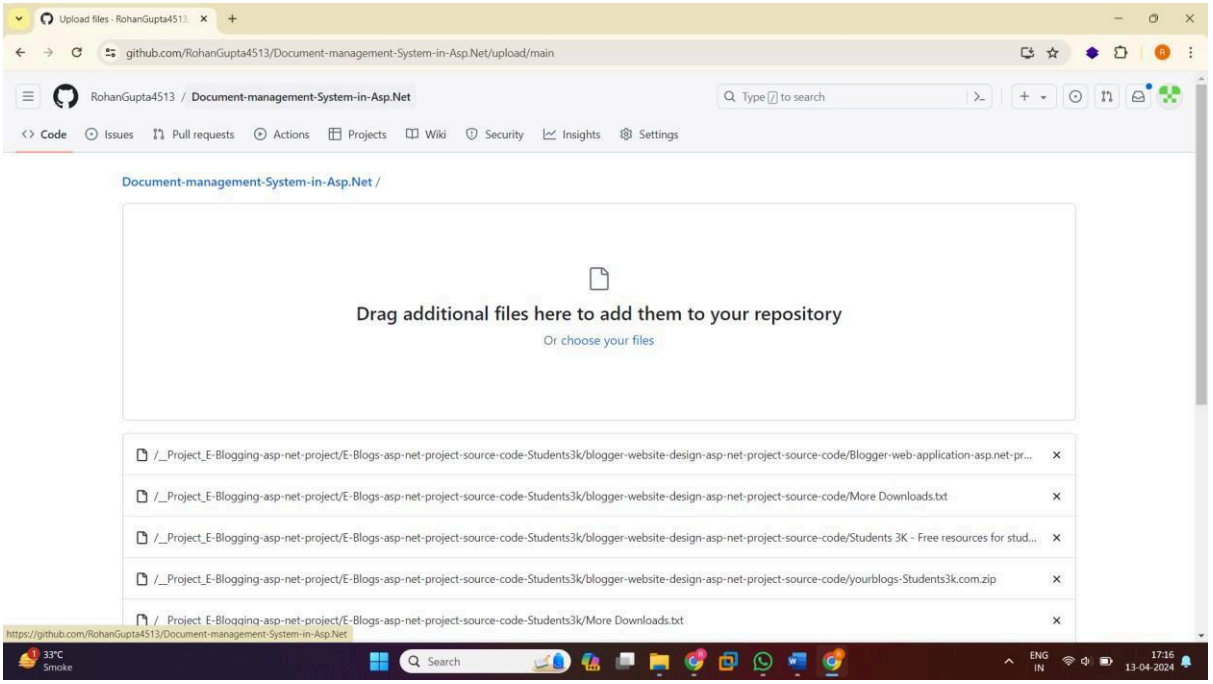


And then click on "Create repository"

Step 4: After successfully creating the repository click on "Add file"
and then on "Upload File"

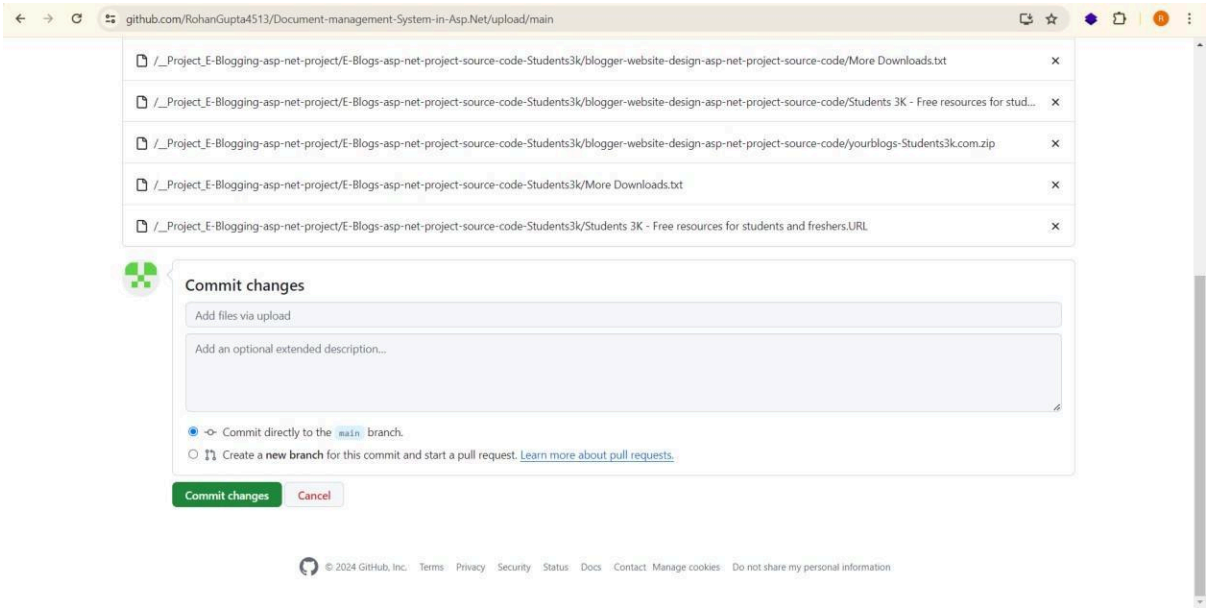


Step 5: Now upload your file here or just simply drag and drop your folder to upload it.

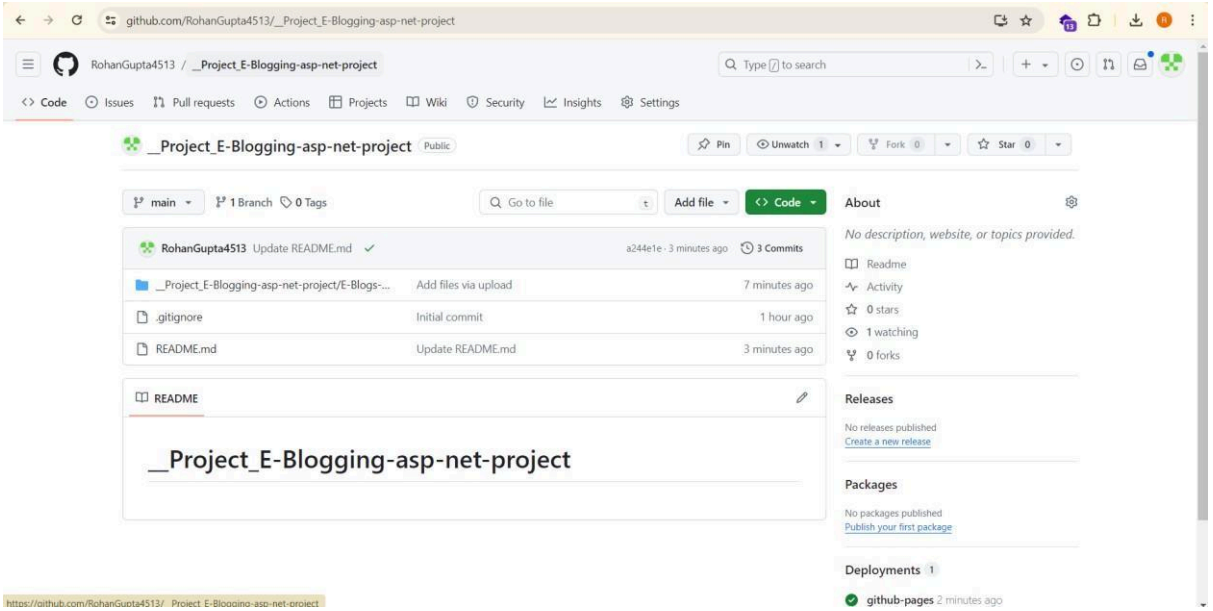


Here we are going to take the project named as "_Project_E-Blogging-asp-net-project"

Step 6: Once the project is uploaded click on "Commit Changes"



Step 7: Here we have successfully uploaded our project.



Practical 5

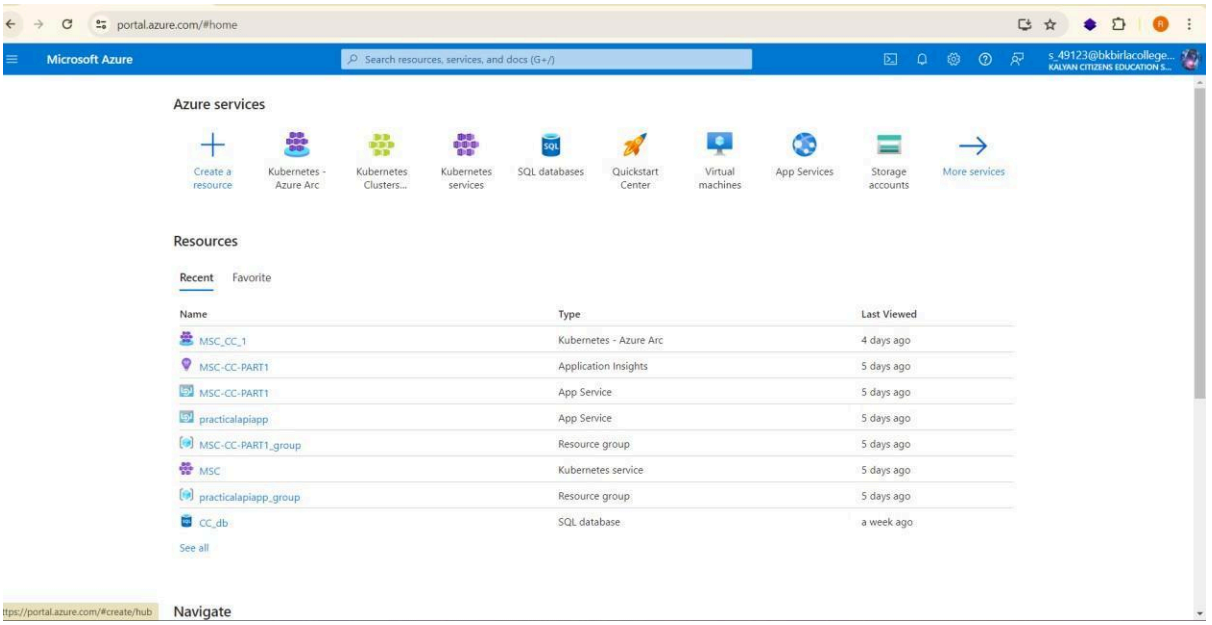
Deploy the GitHub Repository in your Azure API App.

Theory: Set up your GitHub repository: Begin by creating a GitHub repository that houses your API code. If you've already developed your API locally, initialize a Git repository in your project directory
and link it to your GitHub account. You can do this by running git init to initialize a new Git repository and then git remote add origin
<repository_URL> to connect it to your GitHub repository.

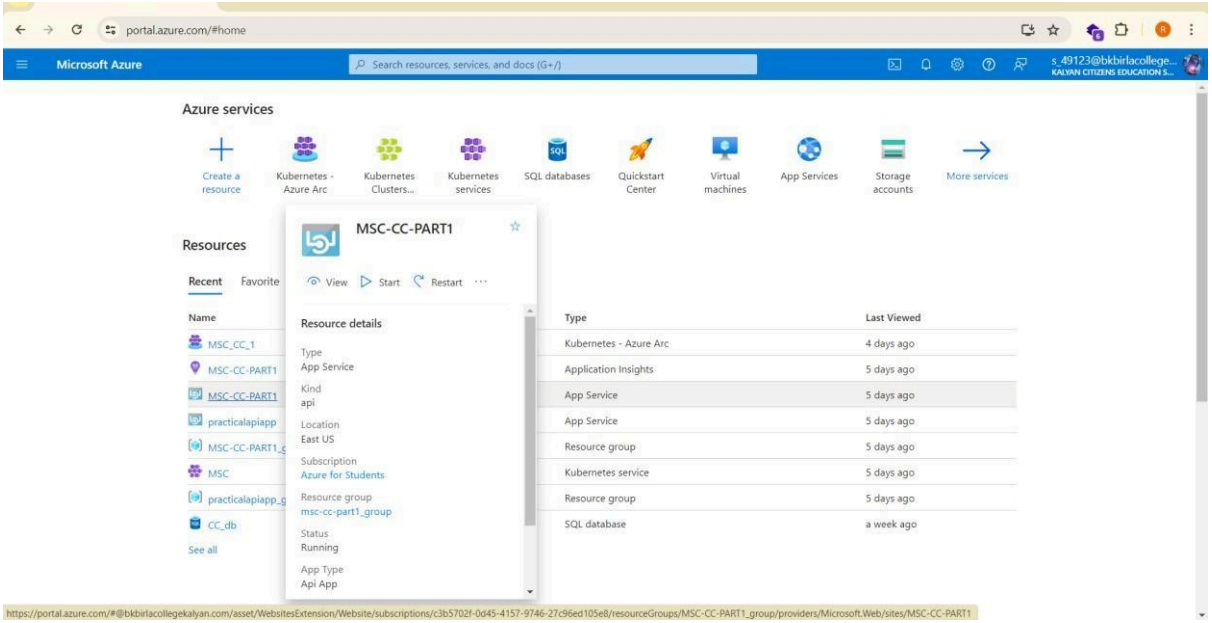
If you're starting fresh, create a new repository on GitHub. Give it a meaningful name and description, and consider adding a README file to provide information about your API. You can also add any necessary files and directories for your API code.

Ensure your repository is properly structured and organized, with
clear documentation and a license if applicable. This foundational step ensures that your codebase is ready for collaboration and deployment.

Step 1: Login to Azure portal with your Azure account.

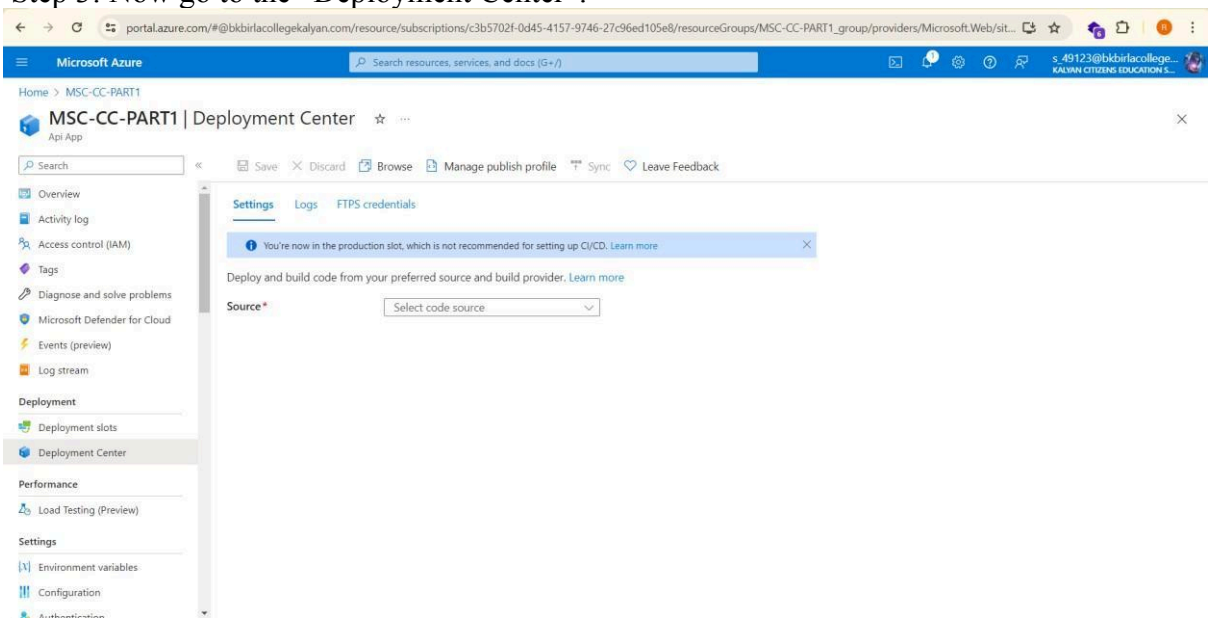


Step 2: Open your API App in which you want to deploy your project which we have created in the previous practical.

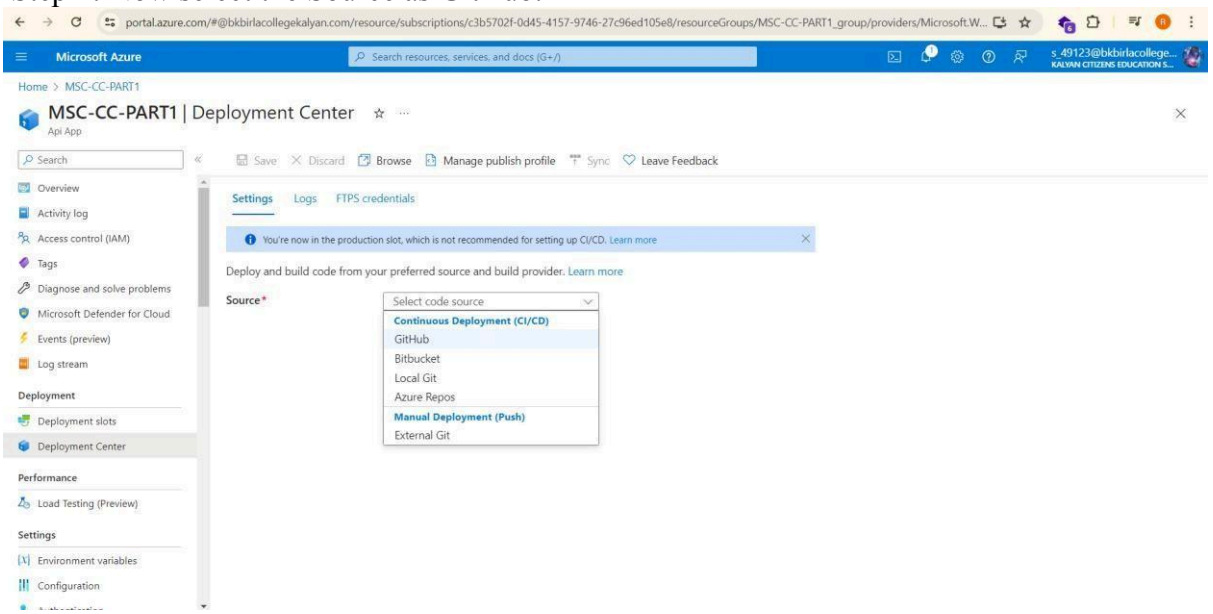


Here we have selected “MSC-CC-PART1” as our API App in which we will be deploying our projects.

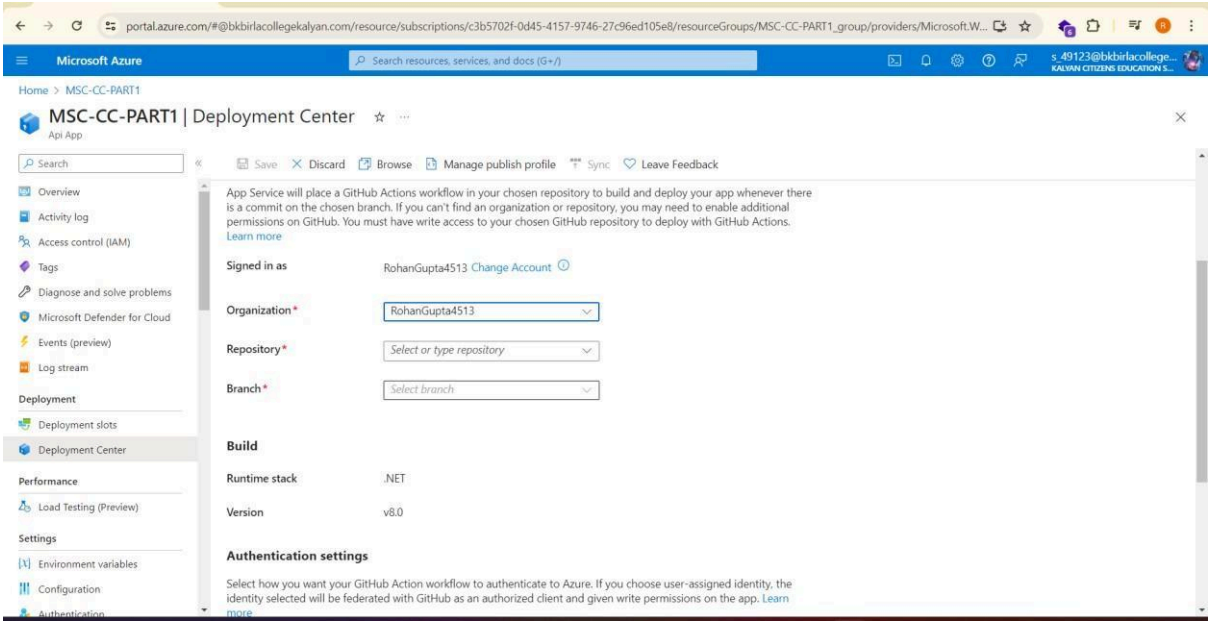
Step 3: Now go to the “Deployment Center”.



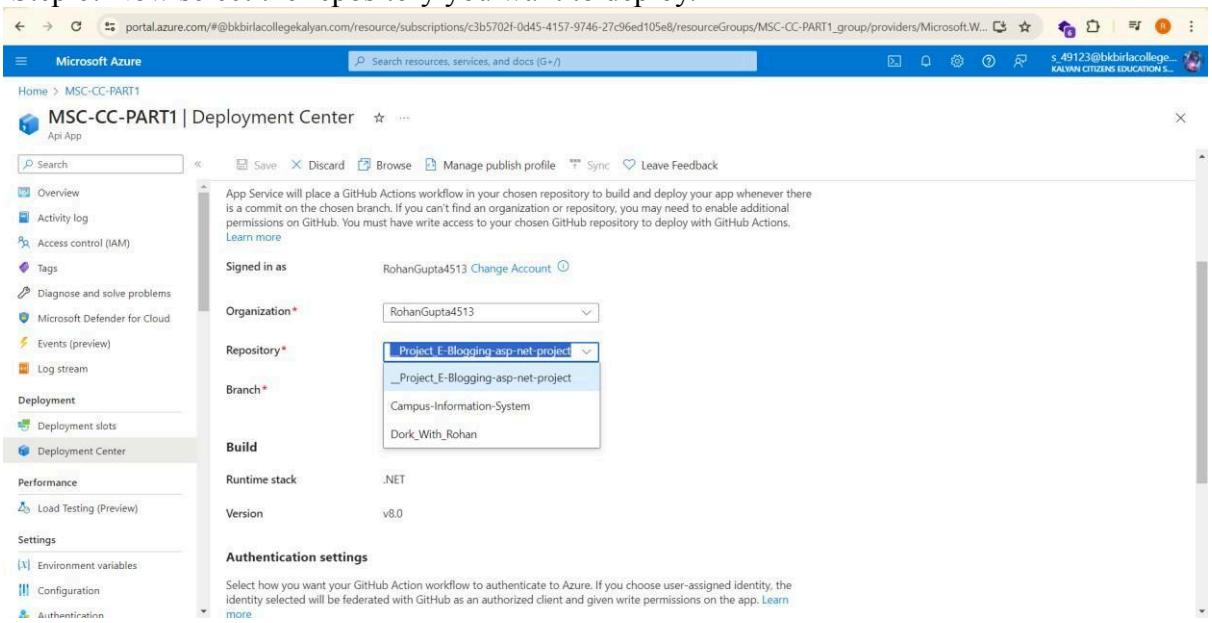
Step 4: Now select the Source as GitHub.



Step 5: Now select your organization as your username of your GitHub account.

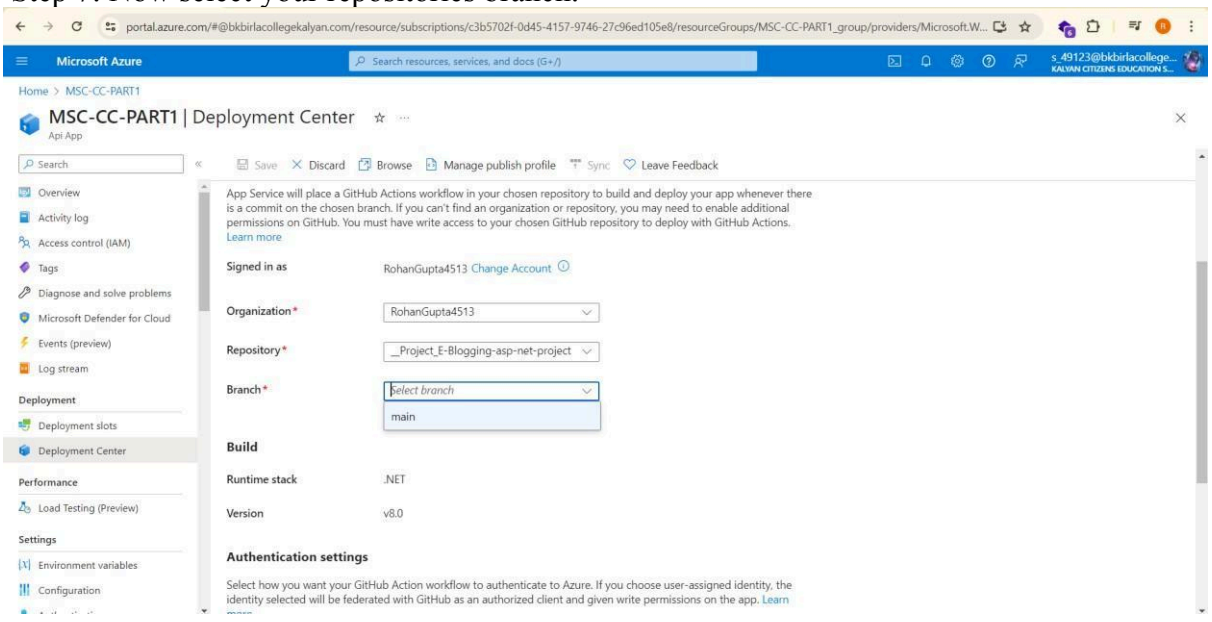


Step 6: Now select the repository you want to deploy.



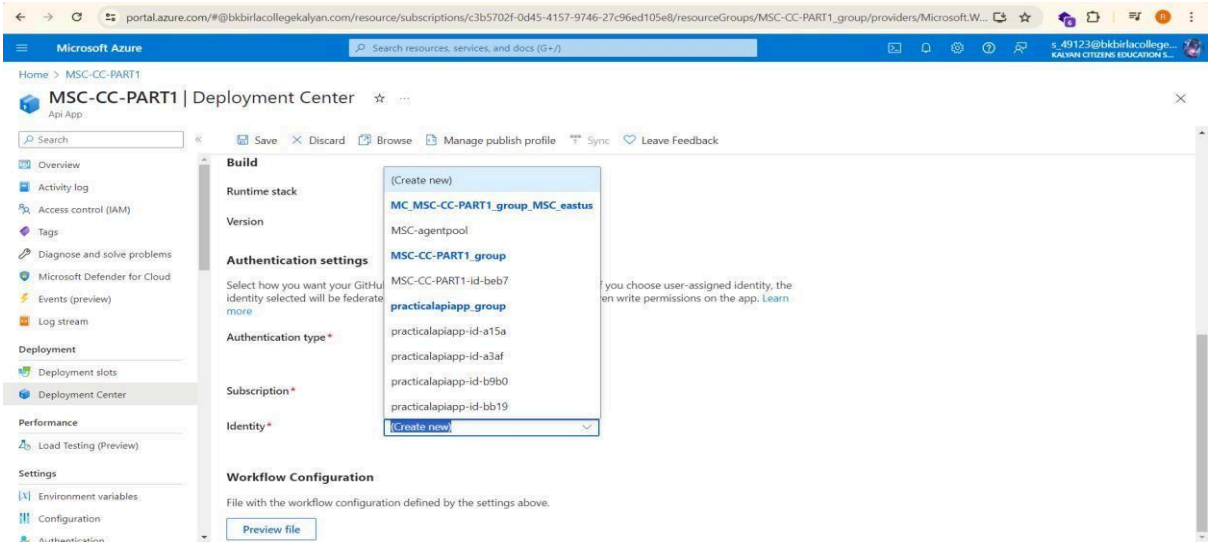
Here we have selected the repository named “ Project_E-Blogging-asp-net-project”.

Step 7: Now select your repositories branch.



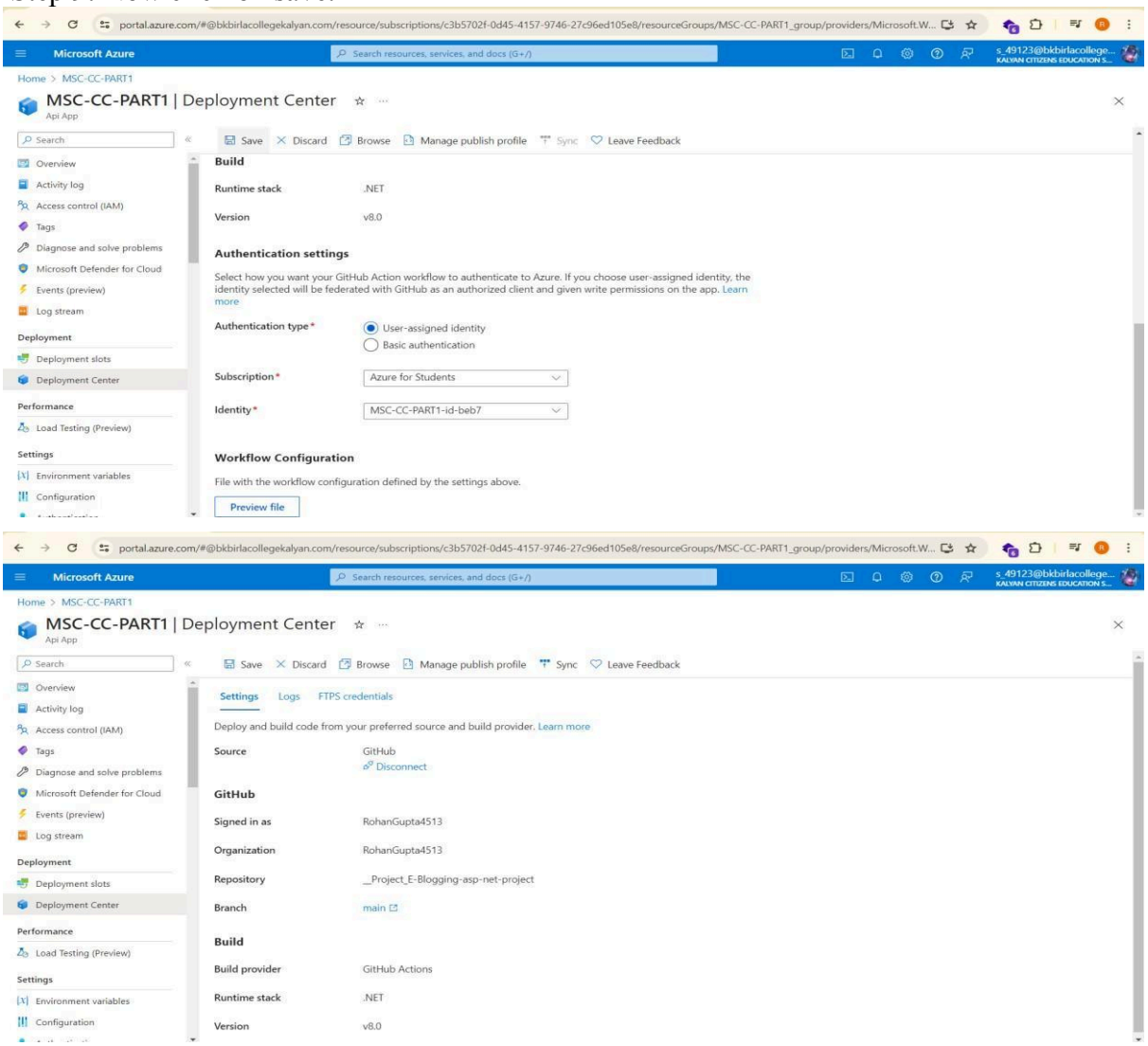
Here we have selected the branch as “main”.

Step 8: Now select the identity of your project or create a new identity of your project.



Here we have selected the identity as “MSC-CC-PART1-id-beb7”

Step 9: Now click on save.

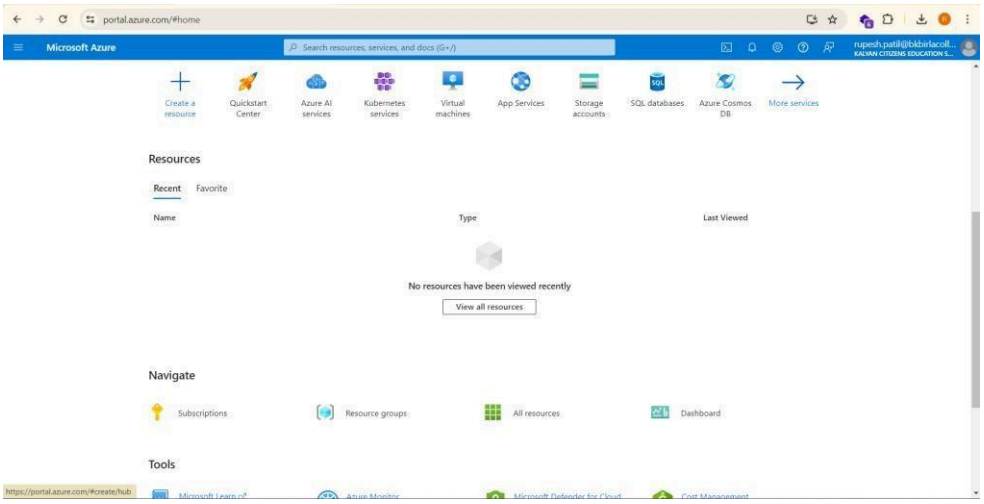


Practical 6

Implement API Authentication Mechanism in your Azure API App

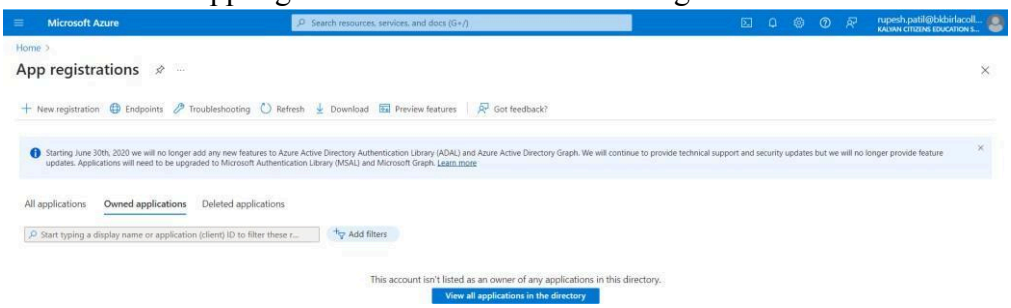
Theory: To implement API authentication in your Azure API App, first, navigate to the Azure Portal and select your API App. Under the "Settings"
section, configure the "Authentication / Authorization" option by turning on the "App Service Authentication" switch. Choose the authentication provider you prefer, such as Azure Active Directory, Facebook, Google, or Twitter. For Azure Active Directory, register your application in the Azure AD to obtain the Client ID and Tenant ID, then configure these in the authentication settings. Set the API to require authentication by selecting the appropriate action under "Action to take when the request is not authenticated." This ensures that every request to your API must include a valid token issued by your chosen provider, effectively securing your API endpoints.

Step 1: Sign in to the Azure portal, navigate to Azure Active Directory in the Azure portal.

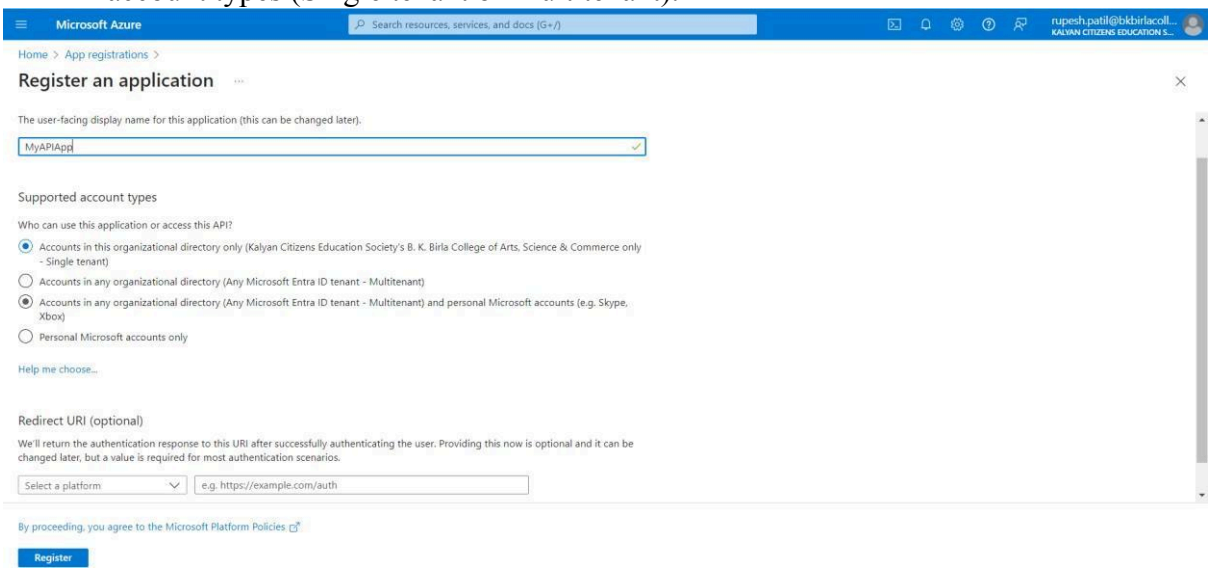


Step 2:

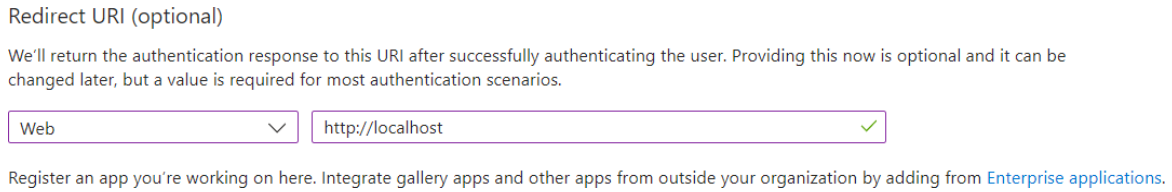
1. Register a new application:
   Go to App registrations and click on New registration.



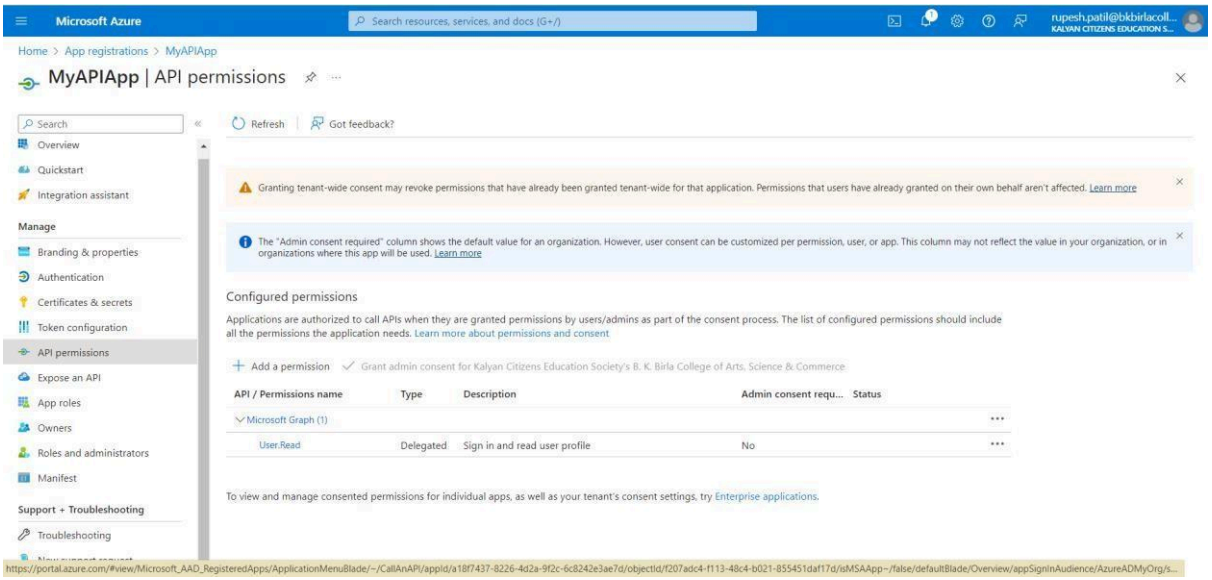
Enter a name for your application (e.g., "MyAPIApp"). Choose the supported account types (Single tenant or Multitenant).



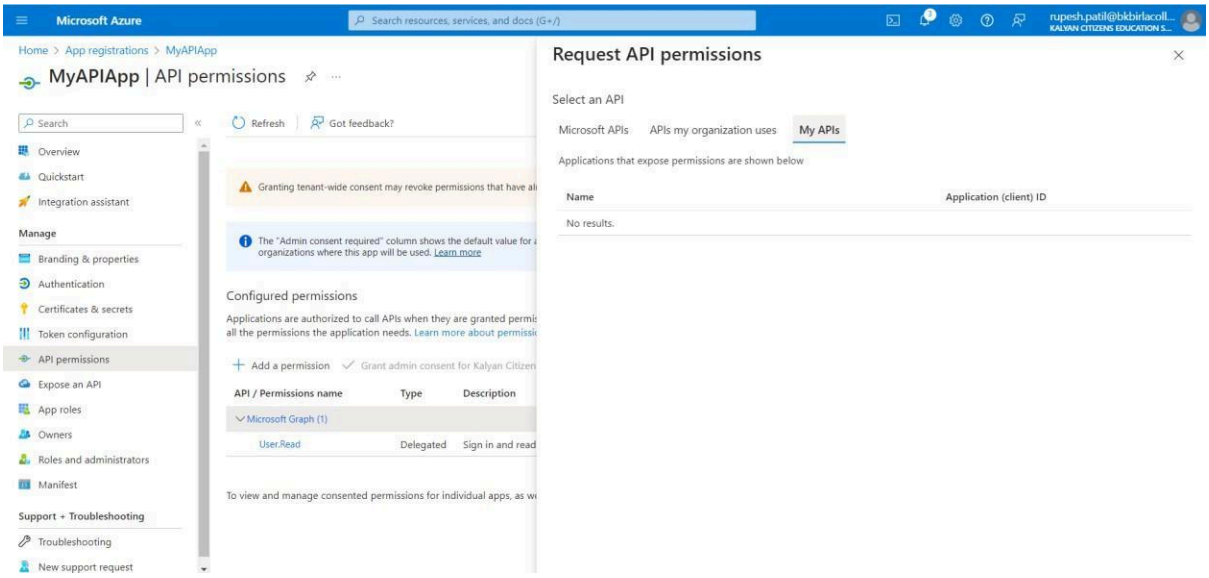
Add a Redirect URI if needed (for testing purposes, you can use http://localhost).



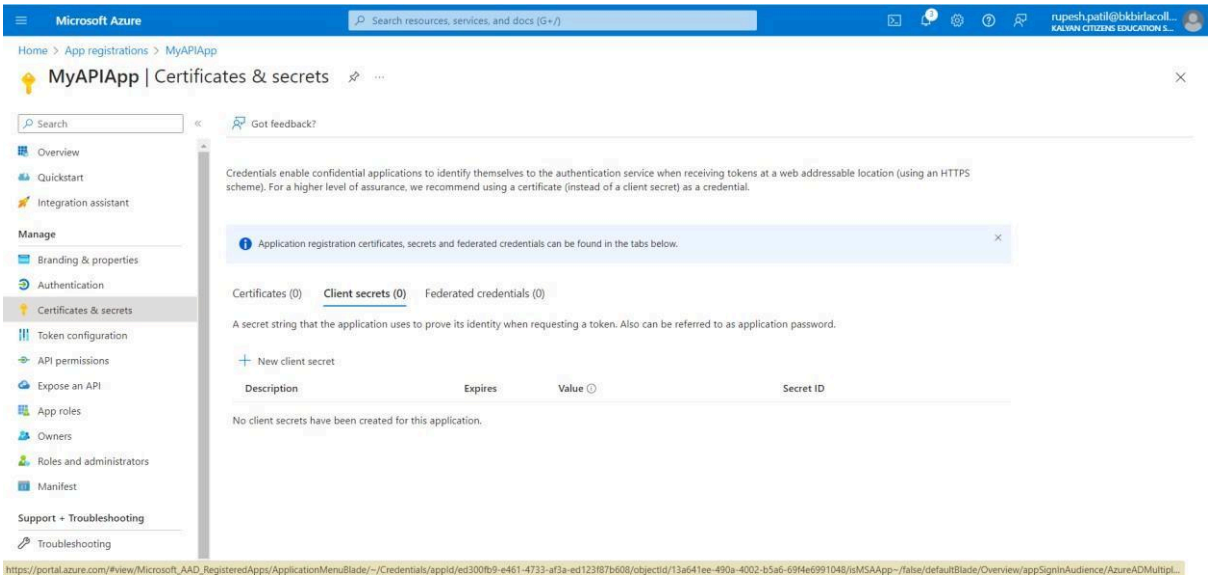
2. Configure API permissions:



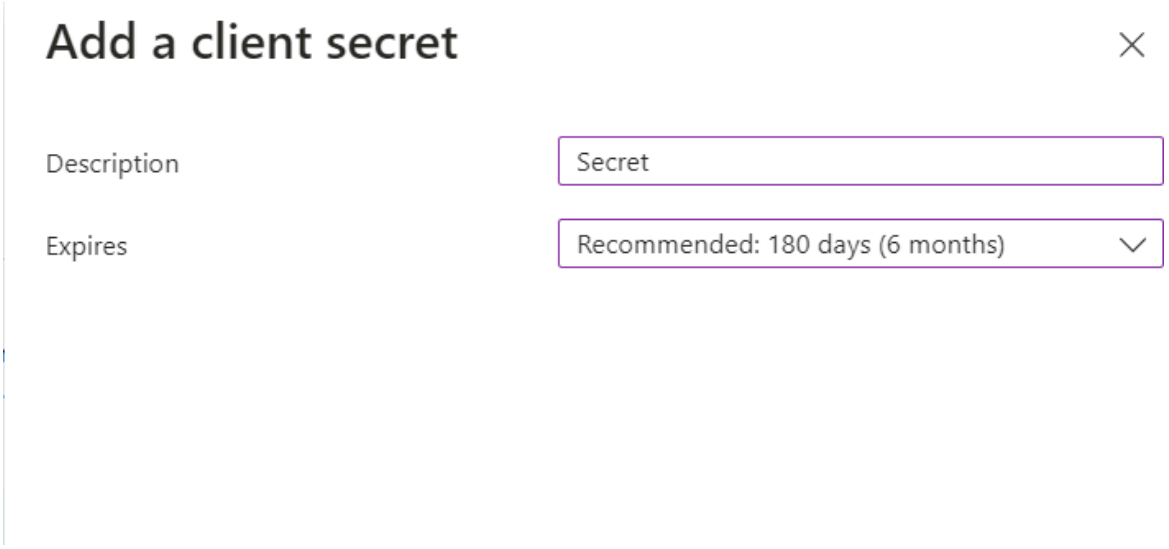
Click on Add permission and select My APIs. Select your API and choose the required permissions.



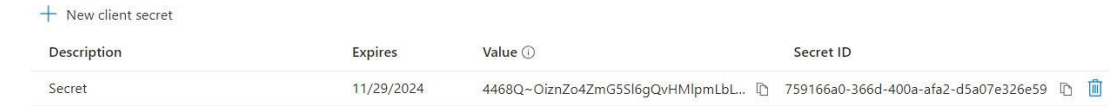
3. Generate a client secret:



Click on New client secret, provide a description, and set an expiration period.

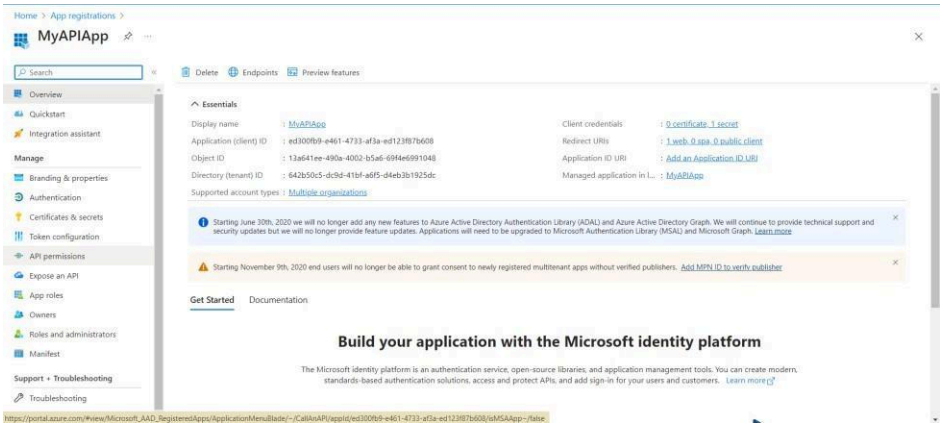


Save the value of the client secret, as it will not be displayed again.



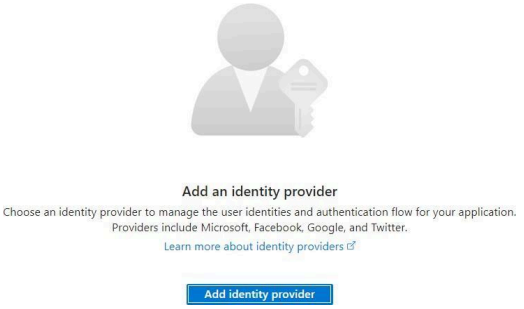
4. Configure Your API App to Use Azure AD Authentication

4.1 Open your API App in the Azure portal.

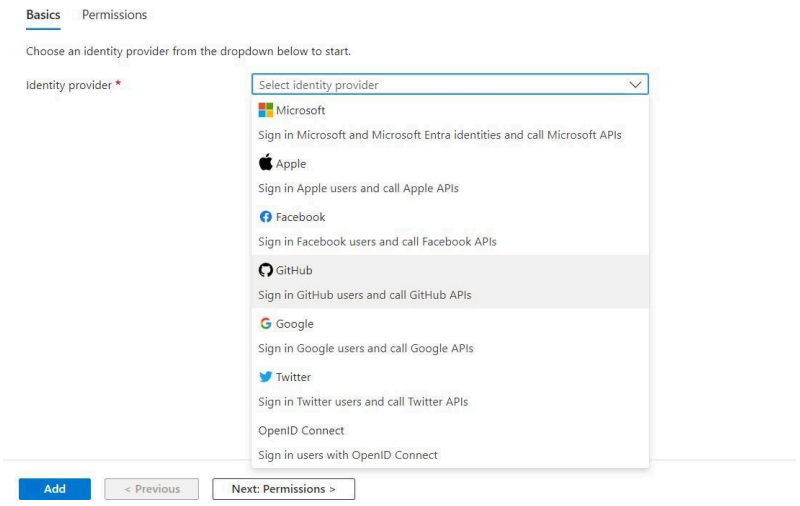


4.2 Navigate to Authentication/Authorization settings:

- Click on Add identity provider.

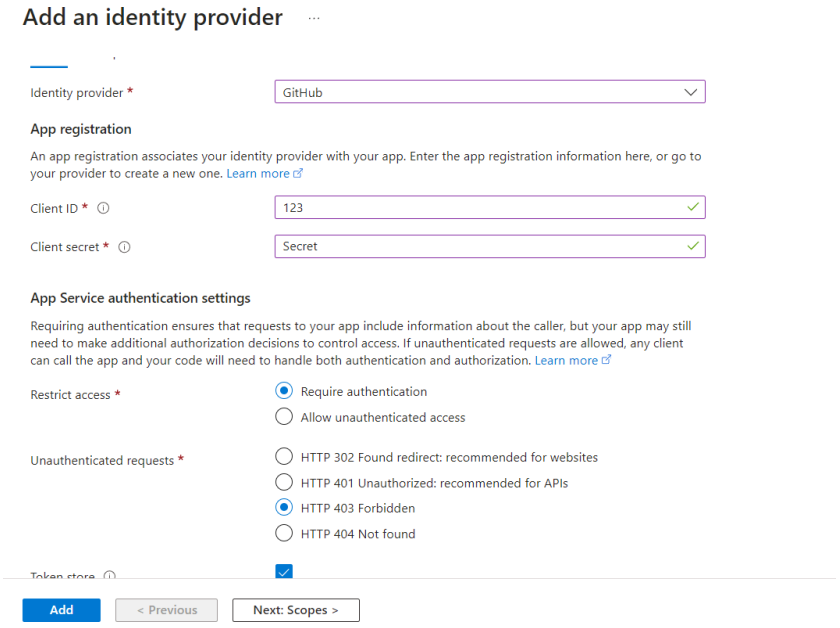


- Now select GitHub as an Identity Provider.



4.3 Configure Azure Active Directory settings:

- Use the Client ID and Client Secret from the Azure AD app registration.



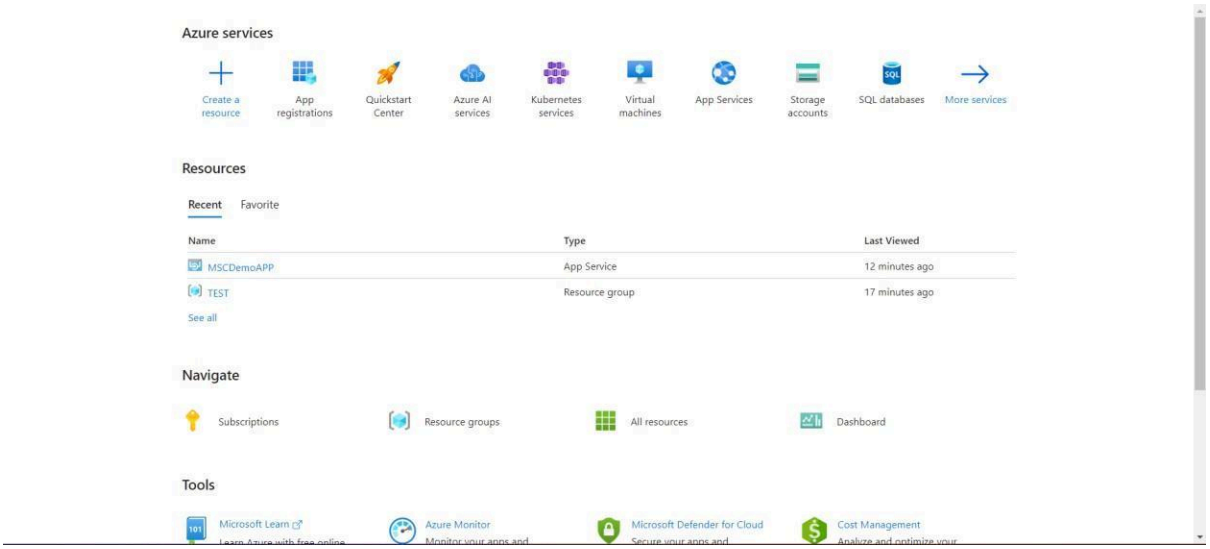
- After clicking on Add your API App will be secured and API Authentication mechanism will be completed successfully.

Practical 7

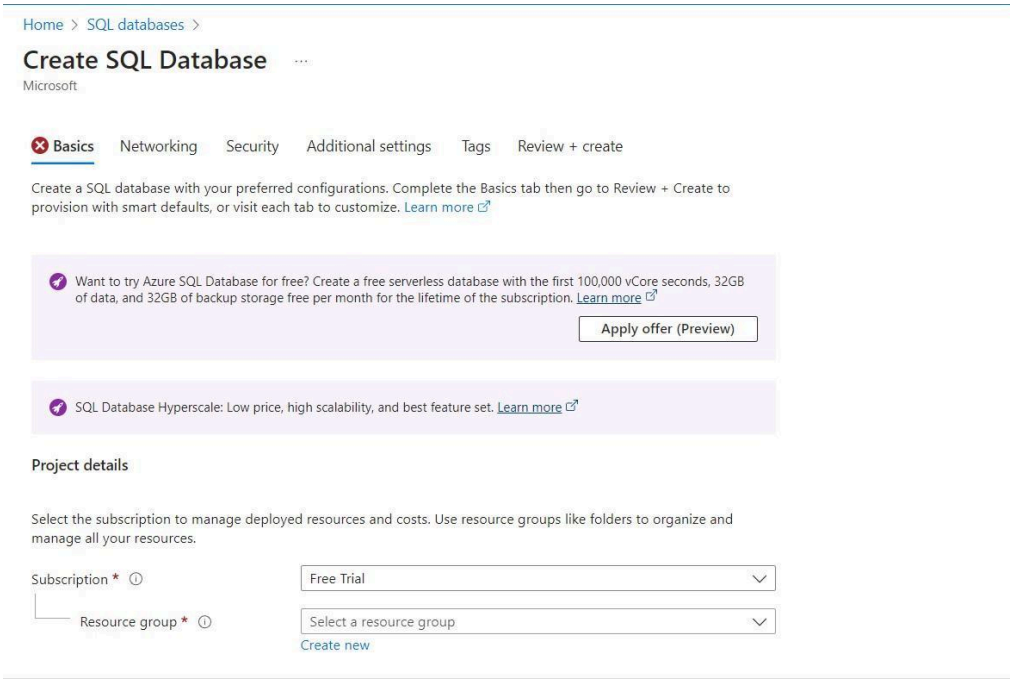
Create a SQL Database and integrate it with your Azure API App

Theory: To create a SQL Database and integrate it with your Azure API App, start by logging into the Azure Portal and navigating to "SQL Databases." Click "Add" to create a new database, providing the necessary details such as database name, server, and pricing tier. Once the database is created, go to your API App in the Azure Portal, and under "Settings," select "Configuration" to add a new connection string. Input the connection details from your SQL Database, including the server name, database name, and authentication credentials. In your API App code, use this connection string to connect to the database, enabling your API endpoints to interact with the SQL Database for data operations. This setup ensures your API can securely access and manage data stored in the SQL Database.

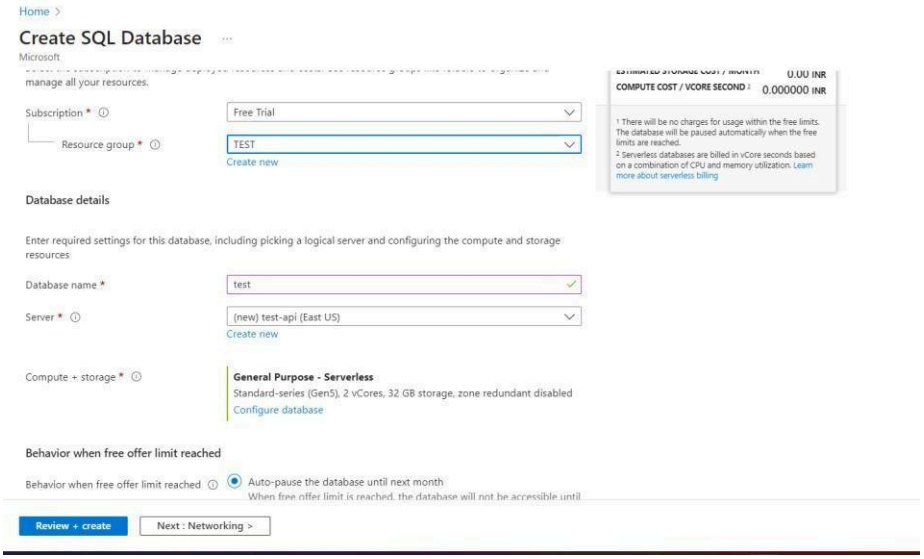
Step 1: Login to your Azure Account in the Azure Portal.



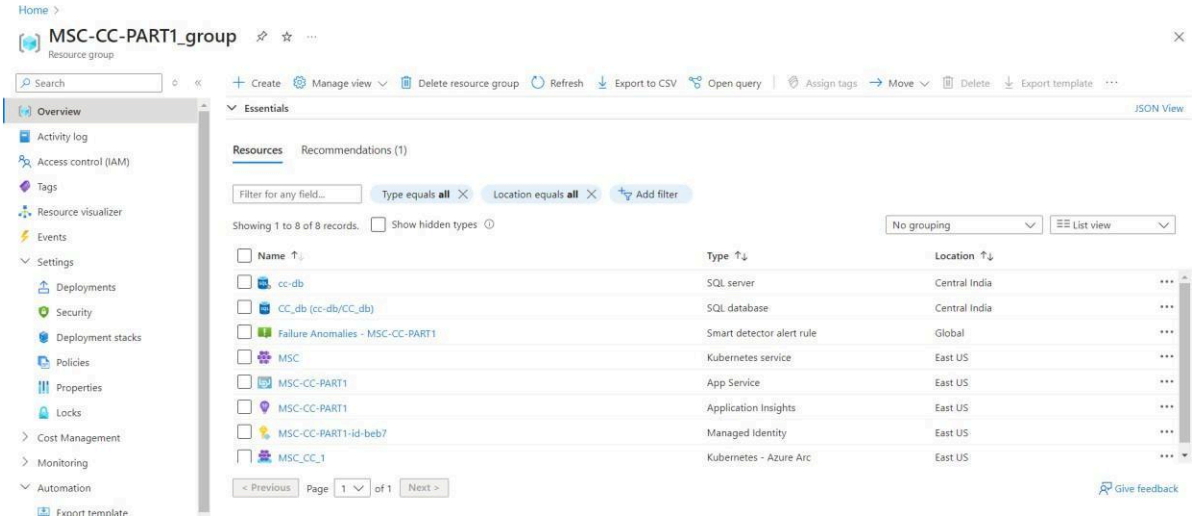
Step 2: Click on SQL Database and then click on Create SQL Database.



Step 3: Fill in all the required details and then click on Review + Create.



Step 4: Once your SQL Database is deployed, it will integrate with your API App since you have added the resource group which is linked with your API App.

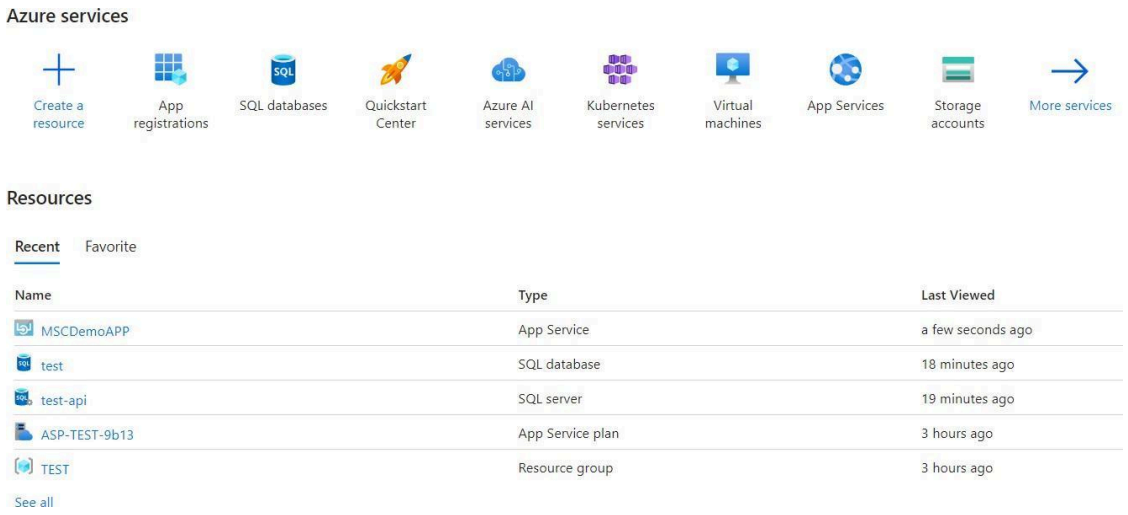


Practical 8

Implement Monitoring and Logging in your Azure API App

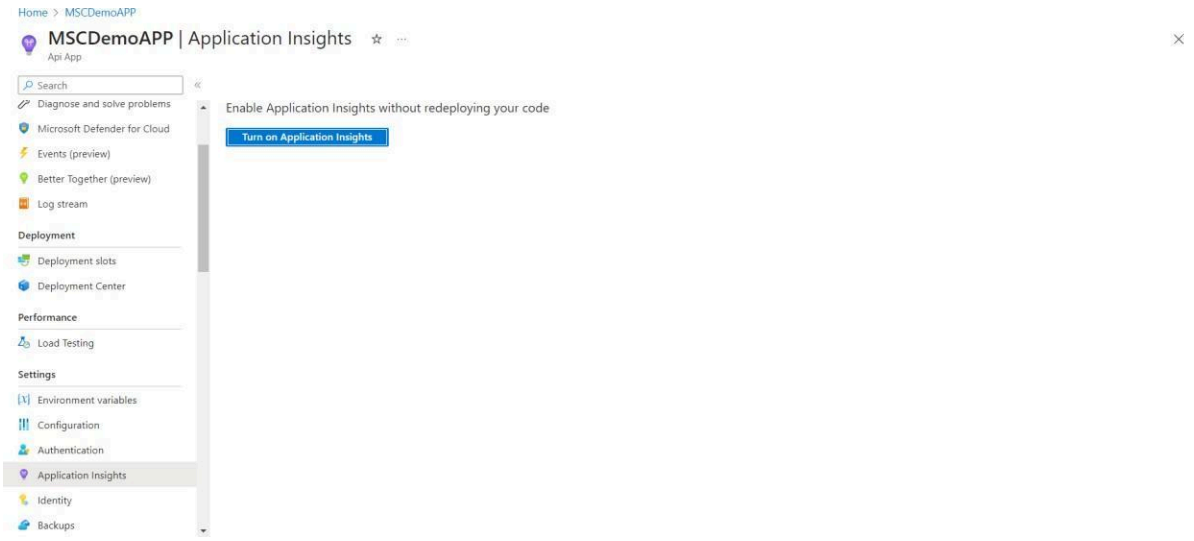
Theory: To implement monitoring and logging in your Azure API App, go to the Azure Portal and select your API App. Under the "Monitoring" section, enable "Application Insights" by selecting it and clicking "Enable." Application Insights provides powerful monitoring and diagnostic capabilities, including request rates, response times, and failure rates. Once enabled, configure it by
adding the Application Insights SDK to your API App code if it's not already integrated. This setup allows you to capture detailed telemetry data.
Additionally, under "Logs," configure "Log Analytics" to centralize logs from your API App. Define diagnostic settings to capture specific logs and metrics, and set up alerts for critical conditions. These tools together offer
comprehensive insights into your API App's performance, usage patterns, and potential issues, facilitating proactive maintenance and troubleshooting.

Step 1: Navigate to your API App in the Azure Portal.



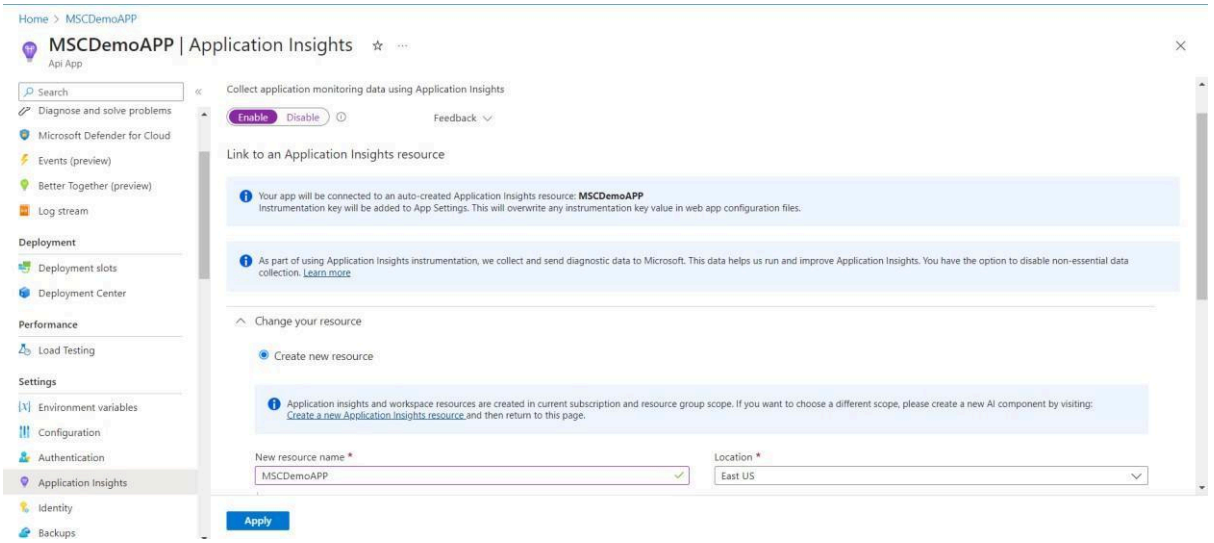
Step 2: Go to Application Insights:

- In the left-hand menu, select Application Insights.
- If Application Insights is not already set up, click Turn on Application Insights and follow the prompts to create a new Application Insights resource or select an existing one.



Step 3: Configure Application Insights:

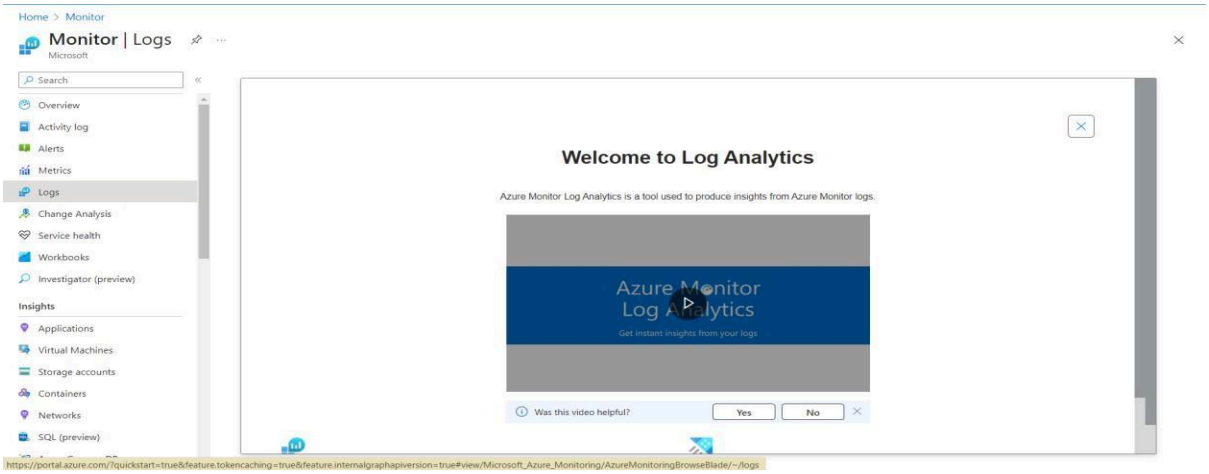
- Once enabled, Application Insights will automatically start collecting data like request rates, response times, failure rates, dependency tracking, and more.



Step 4: Configure Log Analytics

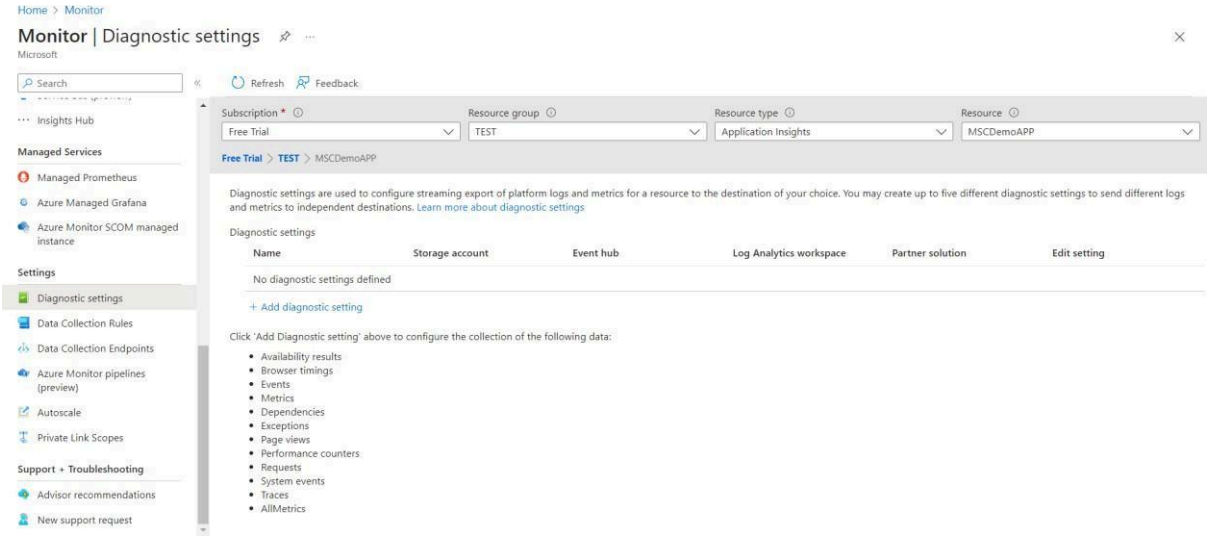
1. Navigate to Azure Monitor:

- In the Azure portal, search for and select Monitor.
- Click on Logs to open the Log Analytics workspace.



Step 5: Link your API App to a Log Analytics workspace:

- If you don't have a workspace, create a new one.
- Navigate back to your API App and go to Diagnostics settings.
- Click Add diagnostic setting, choose the logs and metrics you want to collect, and send them to your Log Analytics workspace.



Step 6: Configure diagnostics logs:

- Select allLogs and any other logs you need.
- Choose the destination (Log Analytics workspace, Storage account, Event Hub).