Meet Agrawal

Assignment - 2 - PoC Document

Basic Setup -

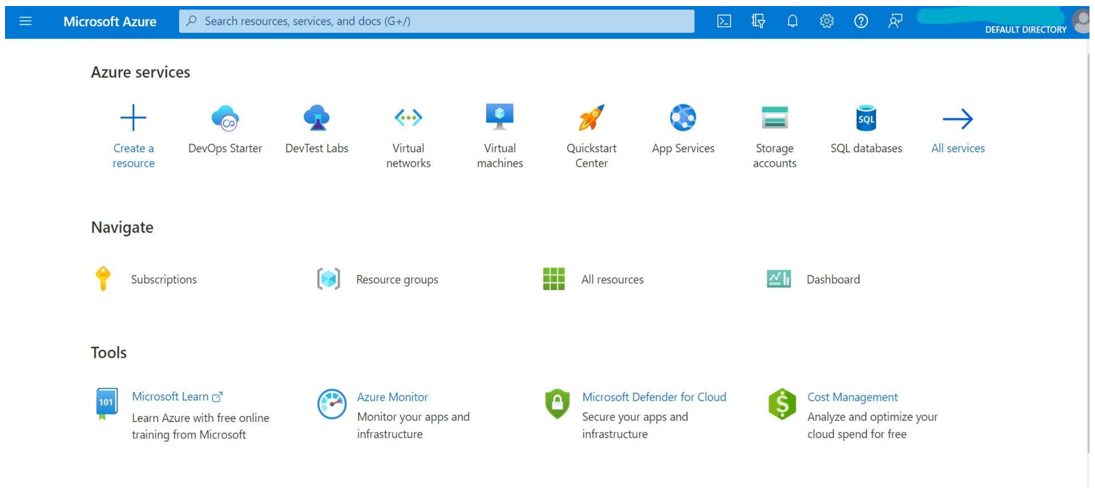
Activating Student Credits for using Azure -

Solve the assignment by activating Azure for Students using your college email id
Use the link - https://azure.microsoft.com/en-in/free/students/

Making account on Azure -

1) Register with a gmail account (not the student account - normal gmail account)
2) Then Click Link above and sign In again
3) You will be redirected to a form where you need to enter your college email
4) Accept confirmation email on your college account and you are all set with the student credits.

Final Screen once done with the setup looks -



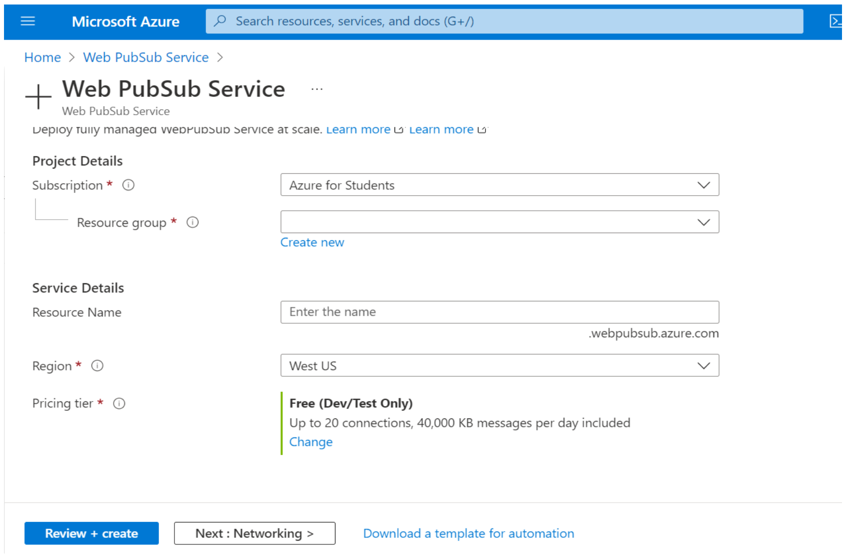
Setting up Node JS :

You need Node JS to solve this assignment
Watch the video to setup if you haven't already done -
https://youtu.be/__7eOCxJyow

Steps to follow :

1) Move to your cloud portal on azure
https://portal.azure.com

2) Create a Azure Web PubSub Service Instance
Remember to change the Pricing Tier to Free (Dev/Test Only) to avoid any payment issues


Home > Web PubSub Service >

Web PubSub Service
Web PubSub Service

Deploy fully managed WebPubSub service at scale. Learn more Learn more

Project Details
Subscription *: Azure for Students
Resource group *:
Create new

Service Details
Resource Name: Enter the name .webpubsub.azure.com
Region *: West US
Pricing tier *: Free (Dev/Test Only)
Up to 20 connections, 40,000 KB messages per day included
Change

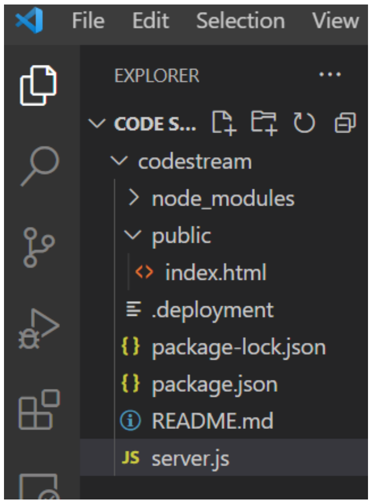
Microsoft Azure
Search resources, services, and docs (G+/)
Review + create
Next : Networking >
Download a template for automation

3) Once the resource is created and deployed, move to Keys and copy the connection string.

4) Now we move towards making our application

File System :

1) Create a CodeStream Folder
2) Run “ `npm init` ” to initialize node in the folder
3) Make a file “ `server.js` ”
4) Run “ `npm i @azure/web-pubsub express` ” to install the relevant package
5) Make a “public” folder containing a file “ `index.html` ”
6) This is the file we’ll serve using our API



5) Using the package for calling the relevant API can be understood using the below link
https://www.npmjs.com/package/@azure/web-pubsub/v/1.0.0

6) Client Streaming can be understood by the below document
https://docs.microsoft.com/en-in/azure/azure-web-pubsub/tutorial-subprotocol?tabs=javascript#set-up-the-project

Code Structure :

1) Initialize an express app in server.js which is used to serve the static folder public
   `const app = express();`
   `app.use(express.static(‘public’))`

2) It consists of a single GET API ‘`/negotiate`’
   This API returns a URL to connect to Web PubSub
3) In case of a Streamer this generates a unique ID using
   `Math.random().toString(36).slice(2,7);`
4) The Client Side consists of 2 roles -
   a) Streamer - writes the code and broadcasts to others
   b) Watcher - watches a streamer’s code

Streamer -

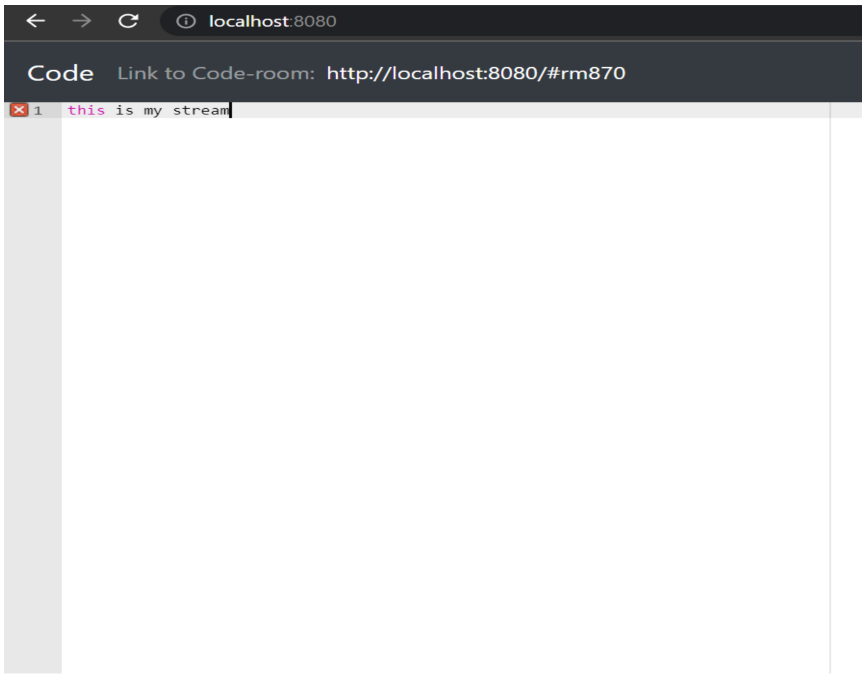
Uses `WebSocket.send()` to send the changes from the code editor (by hooking the `editor.on('change')` event) to a group (whose ID is generated in ‘negotiate’) in Azure Web PubSub. And for performance consideration, it buffers the changes and sends them in a batch every 200 milliseconds.
Implement the logic in - `startStream()`

Watcher -

It receives the changes from Azure Web PubSub and applies them one by one to the code editor (by calling the `applyDelta()` function). Since the change is only a delta from the previous content there needs to be a way to get the full content from streamer when the watcher is connected for the first time. So in this app when the watcher is connected it will send a `sync` message to the streamer (through another group called `{id}-control`) and streamer will send the full content to the group.
Implement the logic in - `watch()`

For Further reference -
https://docs.microsoft.com/en-in/azure/azure-web-pubsub/

Final Output -



```
this is my stream
```

Streamer side -
Hit the URL - http://localhost:8080

A link to Code-room appears - this is the link to be joined in by the watcher to witness the real-time streaming

Testing the Code :

1) Try joining the stream from a different browser window on the same machine
2) Try to join it using a virtual machine - first run the code there with the connection string
   As the sockets are synced in by the cloud the application is available across different machines