# NAME: KSHITIJ GUPTA
# Enrolment Number: 21162101007
# Sub: CS

# Practical – 3[Batch-71]

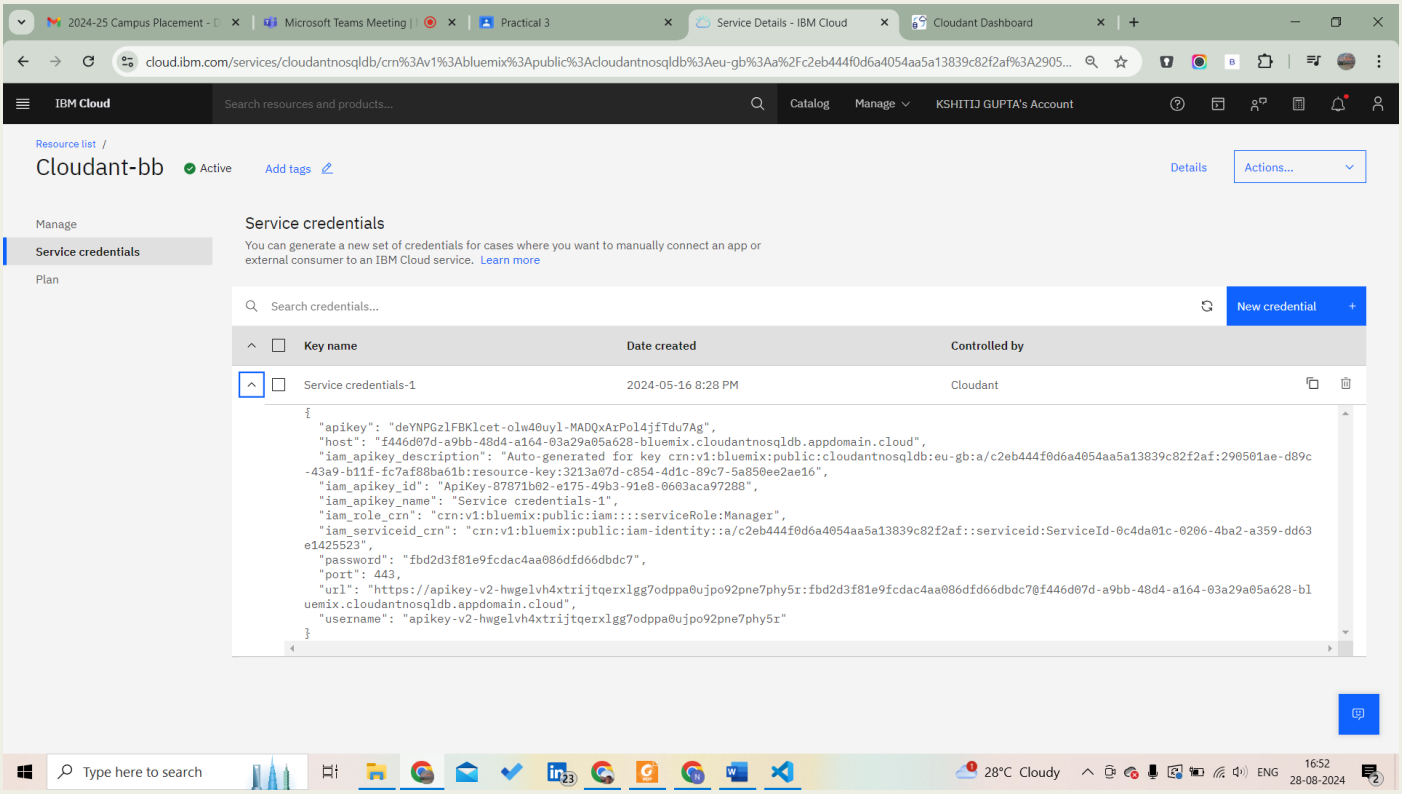
In the first code we are trying to access the cloudant service using the service credentials that are created when we create the cloudant service.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Cloudant CORS Example</title>
</head>
<body>
   <h1>Cloudant CORS Example</h1>
   <button id="fetchDatabases">Fetch Databases</button>
   <pre id="result"></pre>

   <script>
      document.getElementById('fetchDatabases').addEventListener('click', function() {
         const url = 'https://3447fb1b-02ae-4331-923a-607d107471ea-bluemix.cloudantnosqldb.appdomain.cloud/_all_dbs';
         const username = 'apikey-v2-197dnkn3t48agl1wuzpj91l7lo4dkifrzhim8wjf5ykg';
         const password = '0f75c4be5fda84f99a0d4c582ef21b89';
         const headers = new Headers();
         headers.append('Authorization', 'Basic ' + btoa(username + ':' + password));
```

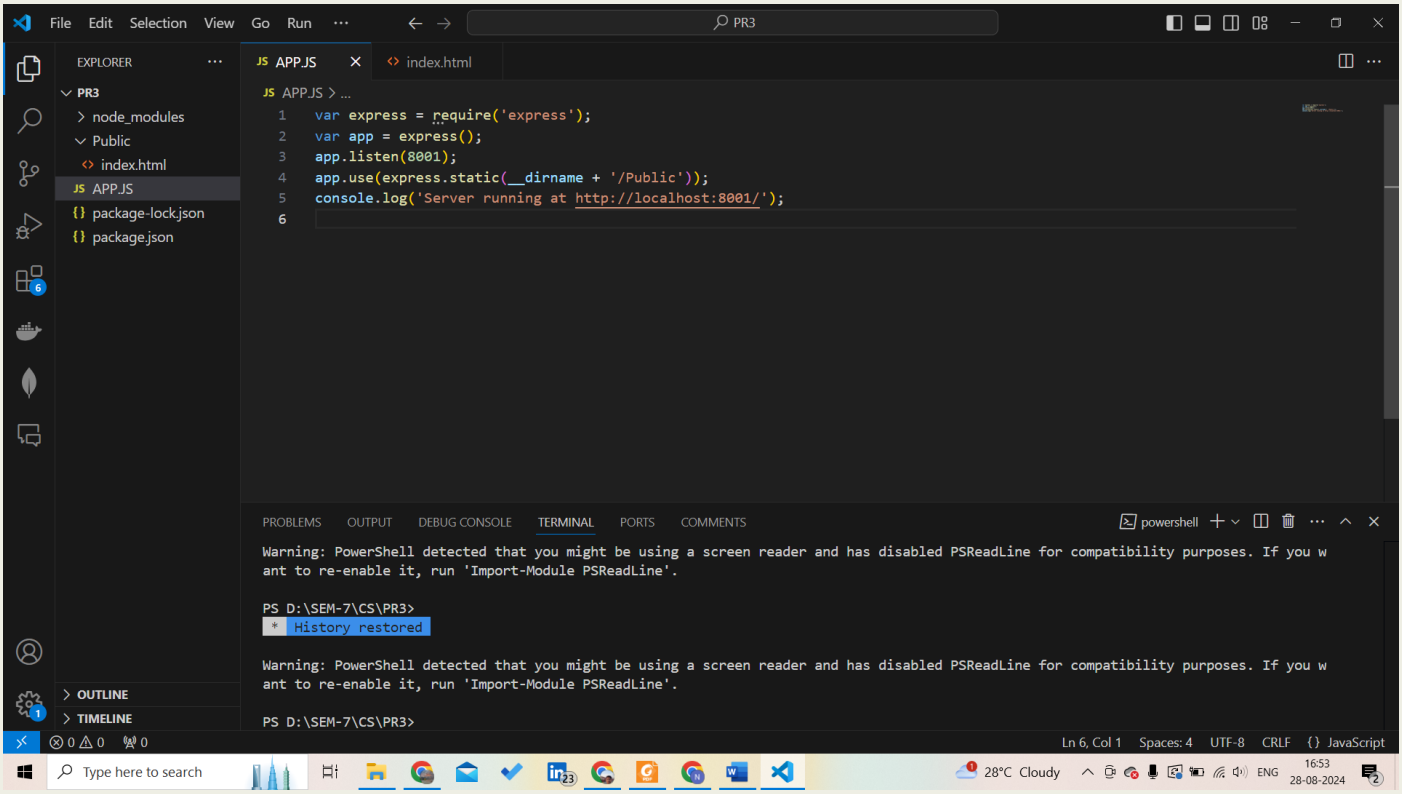
```
        fetch(url, { method: 'GET', headers: headers })
            .then(response => response.json())
            .then(data => {
                document.getElementById('result').textContent = JSON.stringify(data, null, 2);
            })
            .catch(error => {
                document.getElementById('result').textContent = 'Fetch error: ' + error.message;
            });
    });
  </script>
</body>
</html>
```

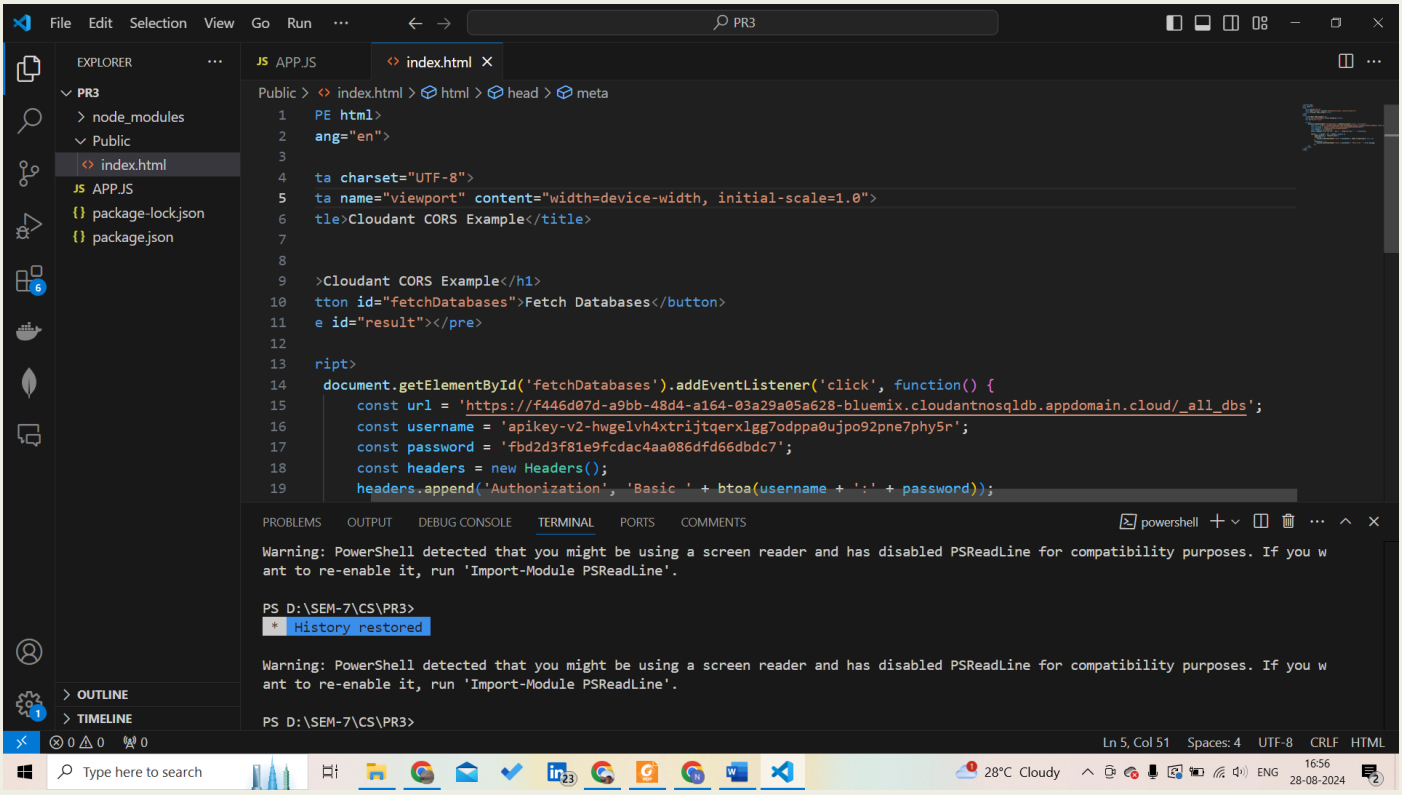
In this script we can see that once the ‘Fetch Databases’ button is clicked it triggers a function, this function makes ‘GET’ request the Cloudant URL, but before that we have Authorization in which username and password are sent with encoding.

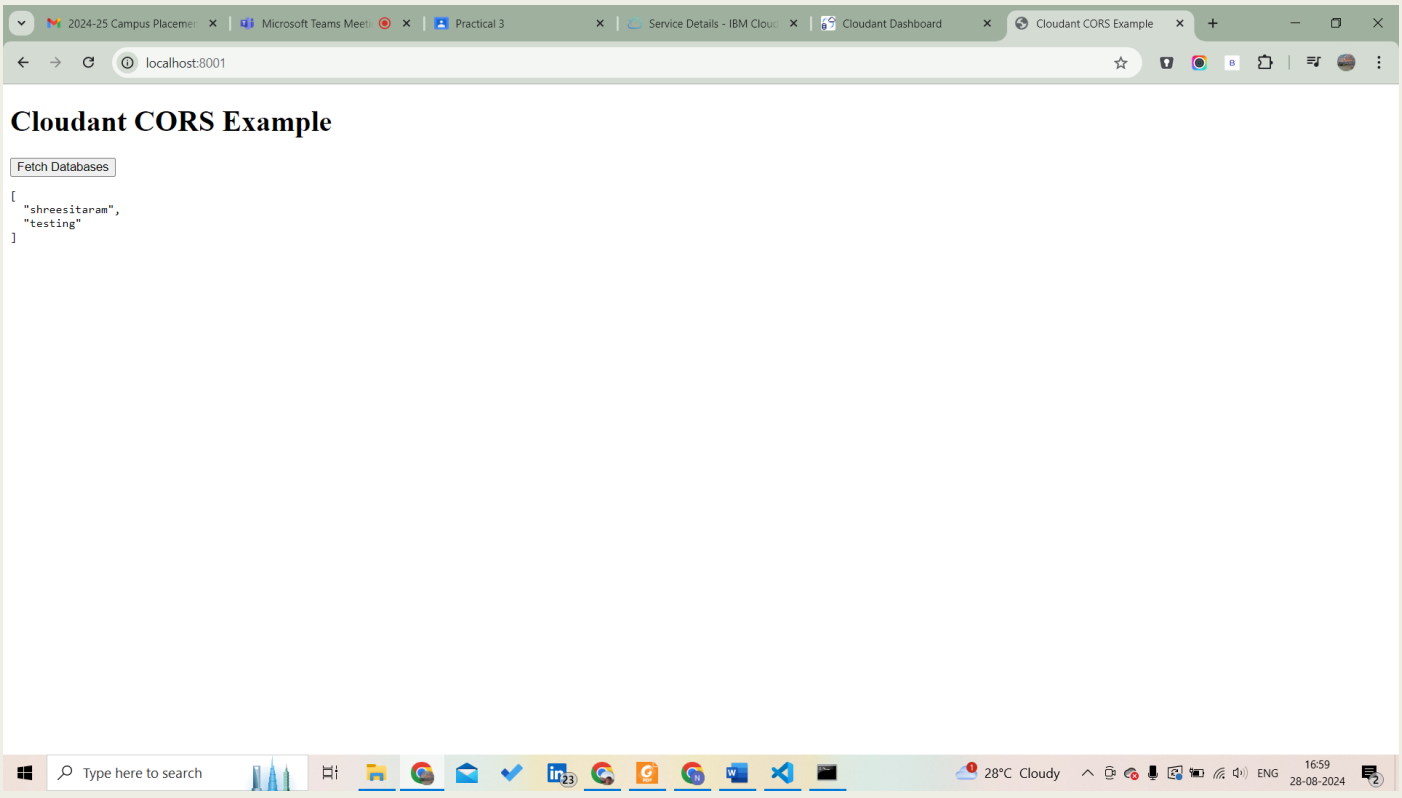
The url, username and password in the code are taken from the Service Credentials in cloudant.



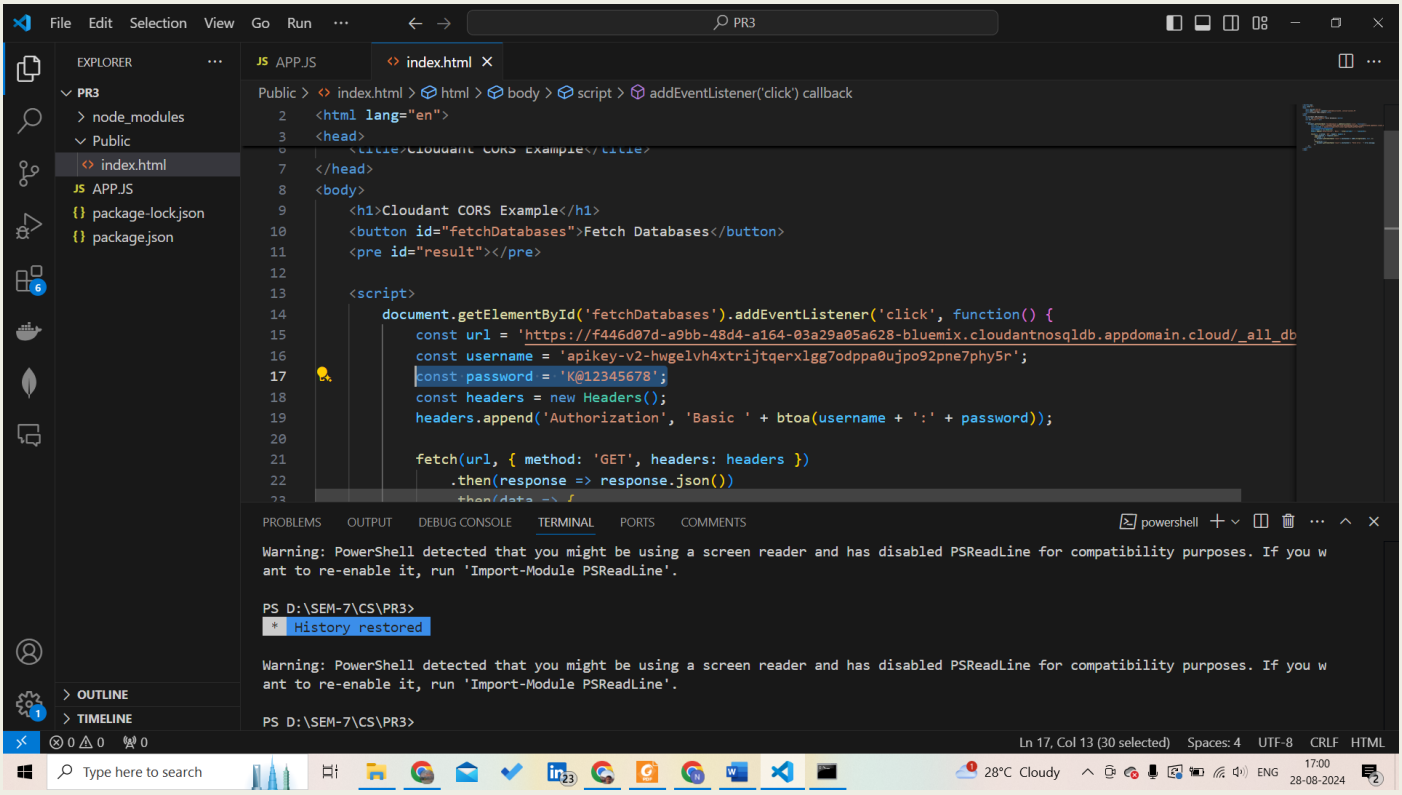

**Now if the username and password are correct then once we run the app.js file we will get the databases.**

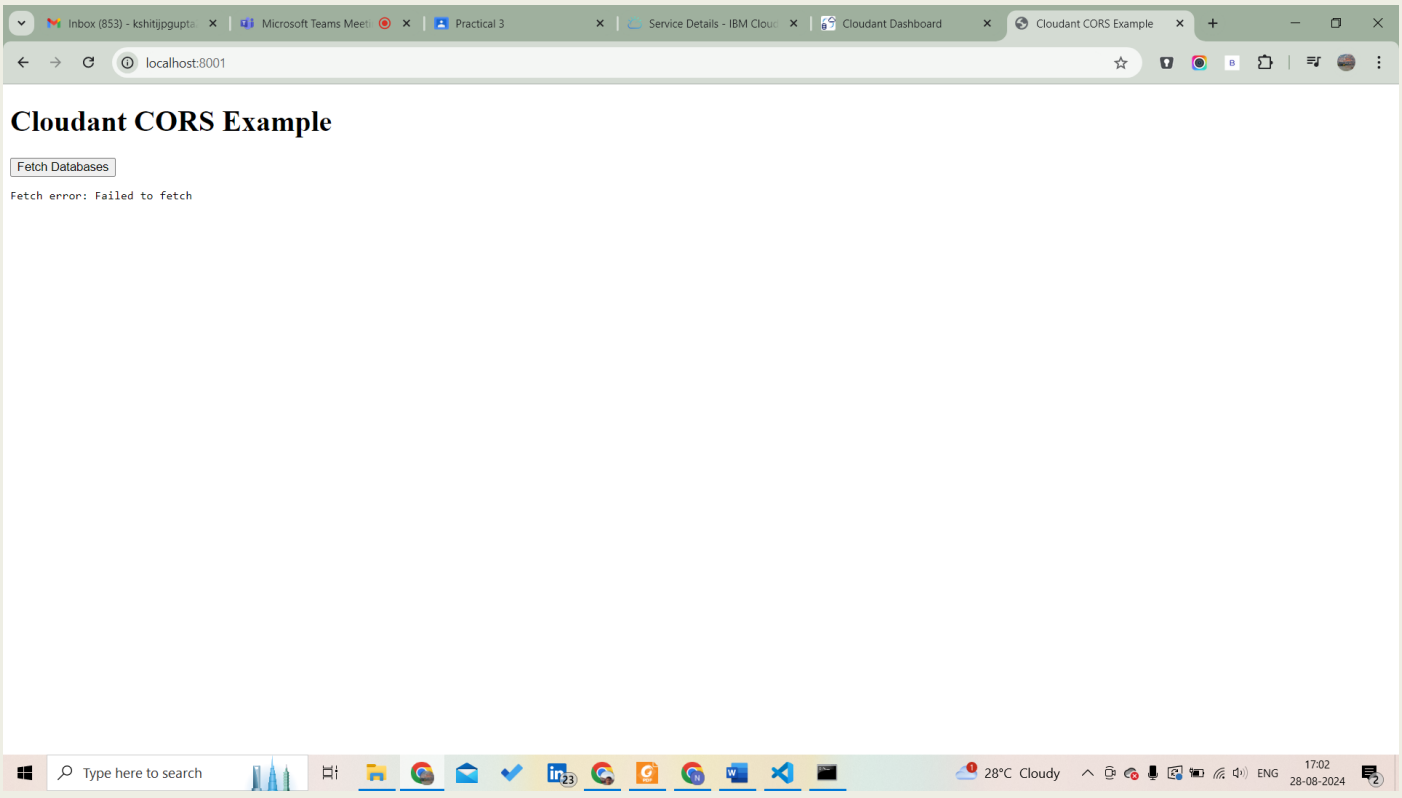




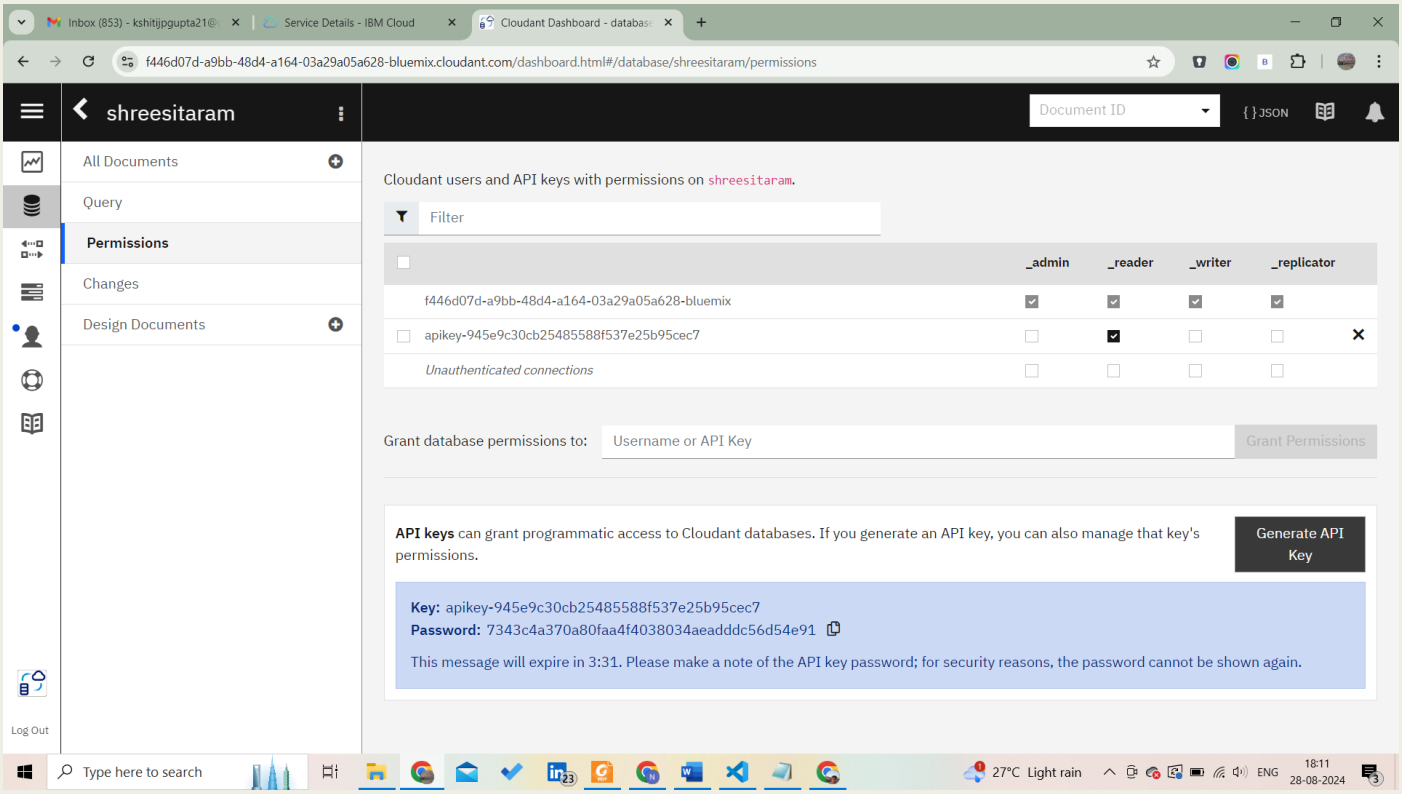

**As we can see we get the list of databases , now if I make changes in my password or username I will not be able to get the list.**




**Now we will try to access a particular database by generating an API.**

**Now to generate an API first we go to the Cloudant dashboard and then in the Permissions tab in the database, there we get an option to generate an API key.**



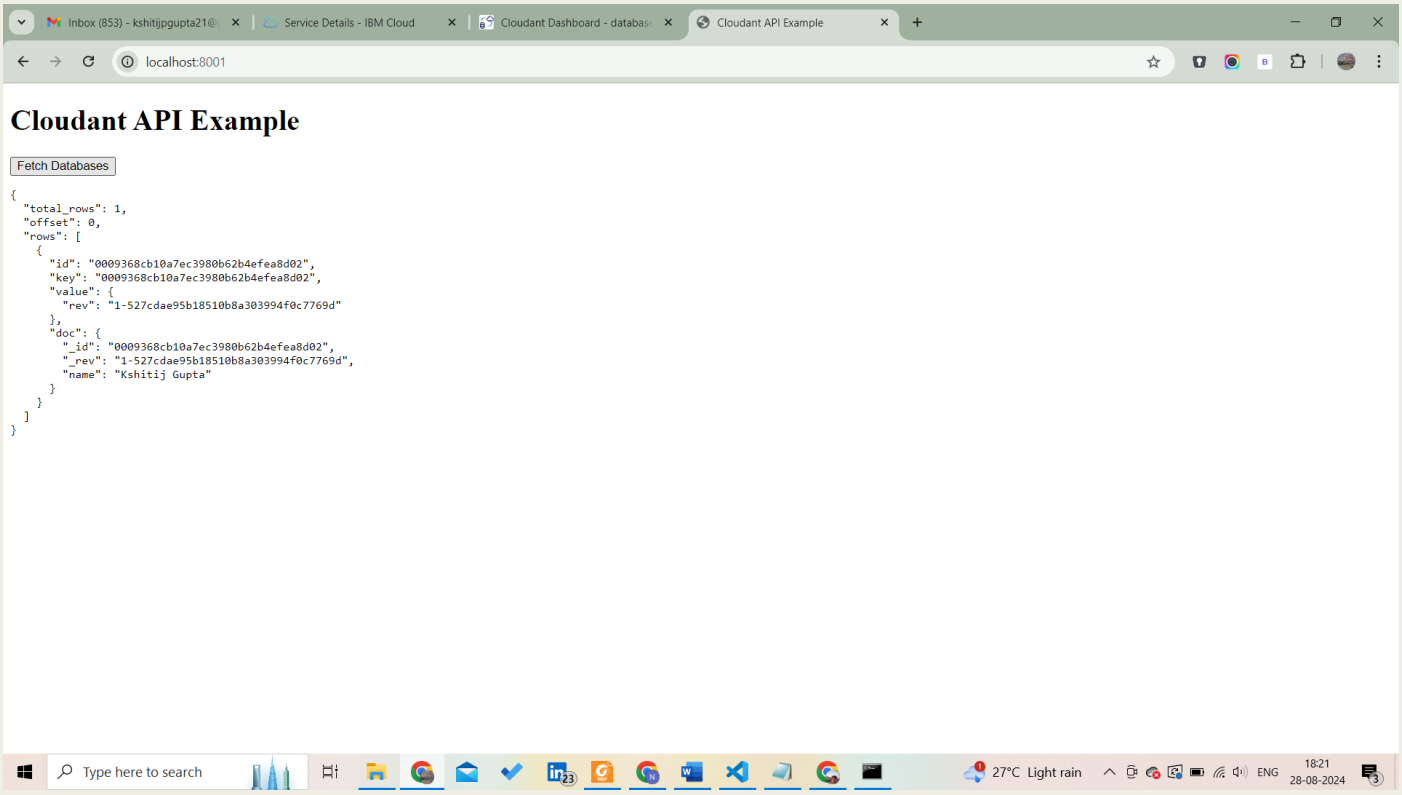

**Here I have given the reader permission to the API key.**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cloudant API Example</title>
</head>
<body>
    <h1>Cloudant API Example</h1>
    <button id="fetchData">Fetch Databases</button>
    <pre id="result"></pre>

    <script>
        document.getElementById('fetchData').addEventListener('click', function() {
    const apiKey = 'a945e9c30cb25485588f537e25b95cec7';
const apiPassword = '7343c4a370a80faa4f4038034aeadddc56d54e91';
```

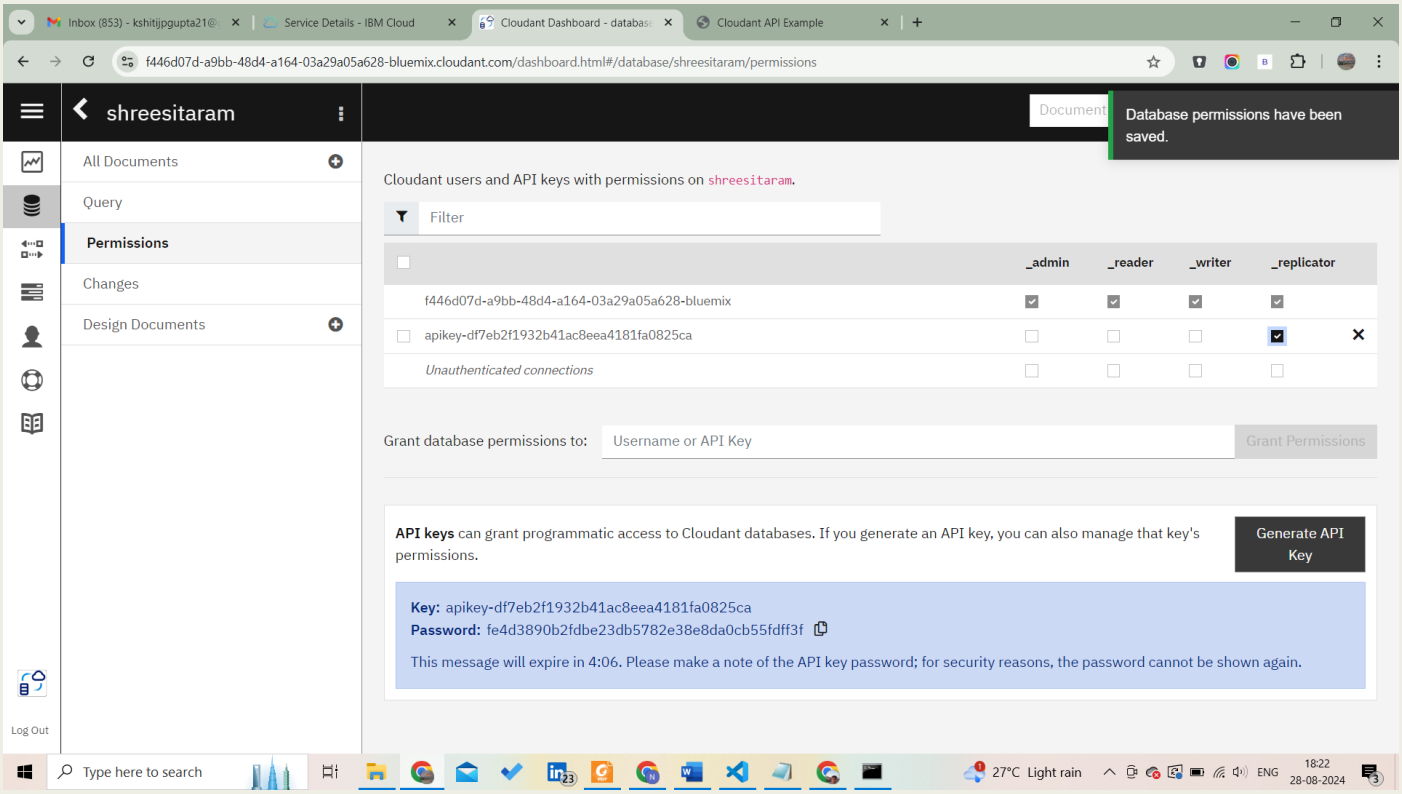
```
const url = 'https://f446d07d-a9bb-48d4-a164-03a29a05a628-bluemix.cloudantnosqldb.appdomain.cloud/shreesitaram/_all_docs?include_docs=true';


const headers = new Headers();

headers.append('Authorization', 'Basic ' + btoa(apiKey + ':' + apiPassword));


fetch(url, { method: 'GET', headers: headers })

        .then(response => response.json())

        .then(data => {

            document.getElementById('result').textContent = JSON.stringify(data, null, 2);

        })

        .catch(error => {

            document.getElementById('result').textContent = 'Fetch error: ' + error.message;

        });


    });

  </script>

</body>

</html>
```

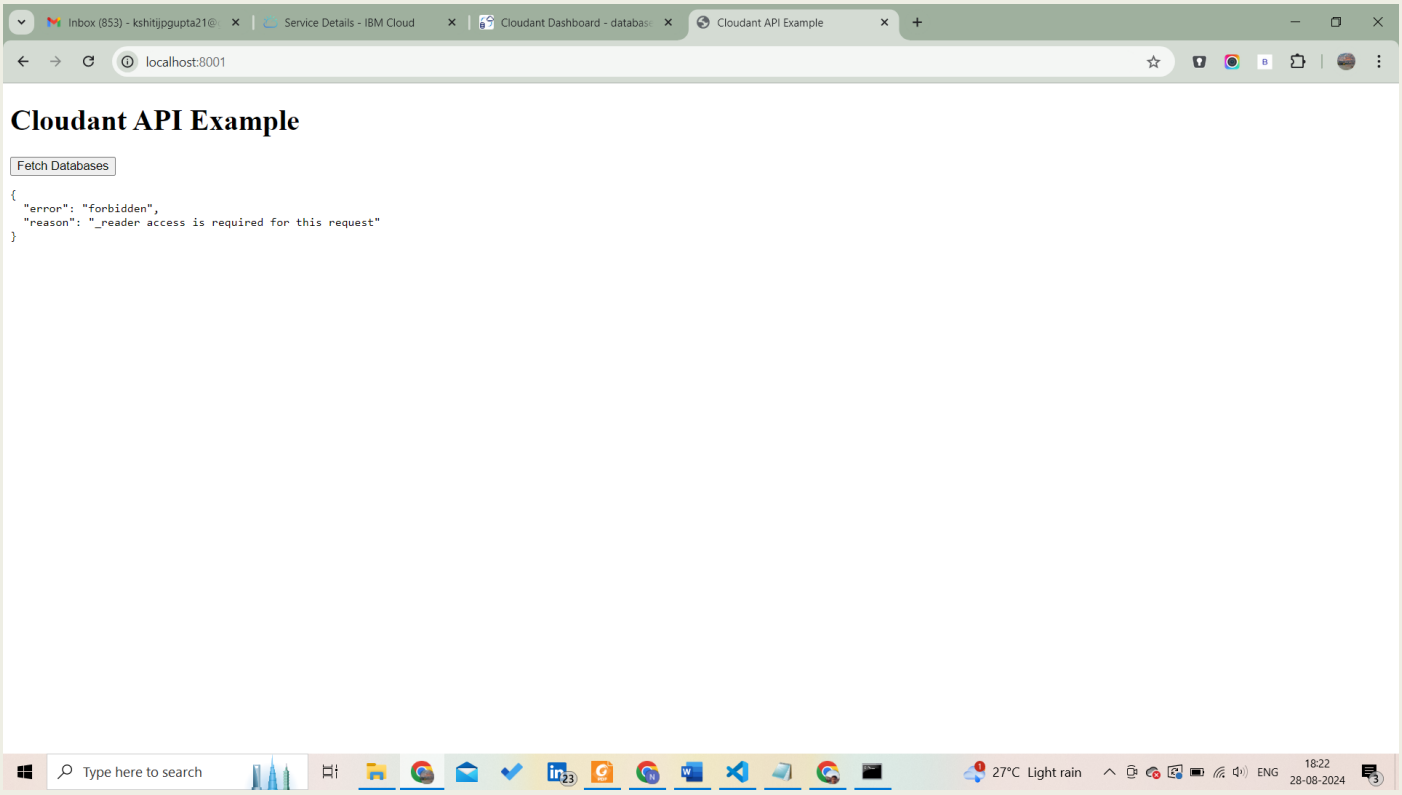
Here the URL is the same hostname we had earlier but this time along with the database name we want to access. Also this time the authentication is done using the API key and the password we generated is also encoded using btoa() function.

As we can see that after we run the code we are able to see the documents in the database. It is possible because we gave the reader access.




**Now if I change the permission and do not give the reader access. We will not be able to fetch the documents.**

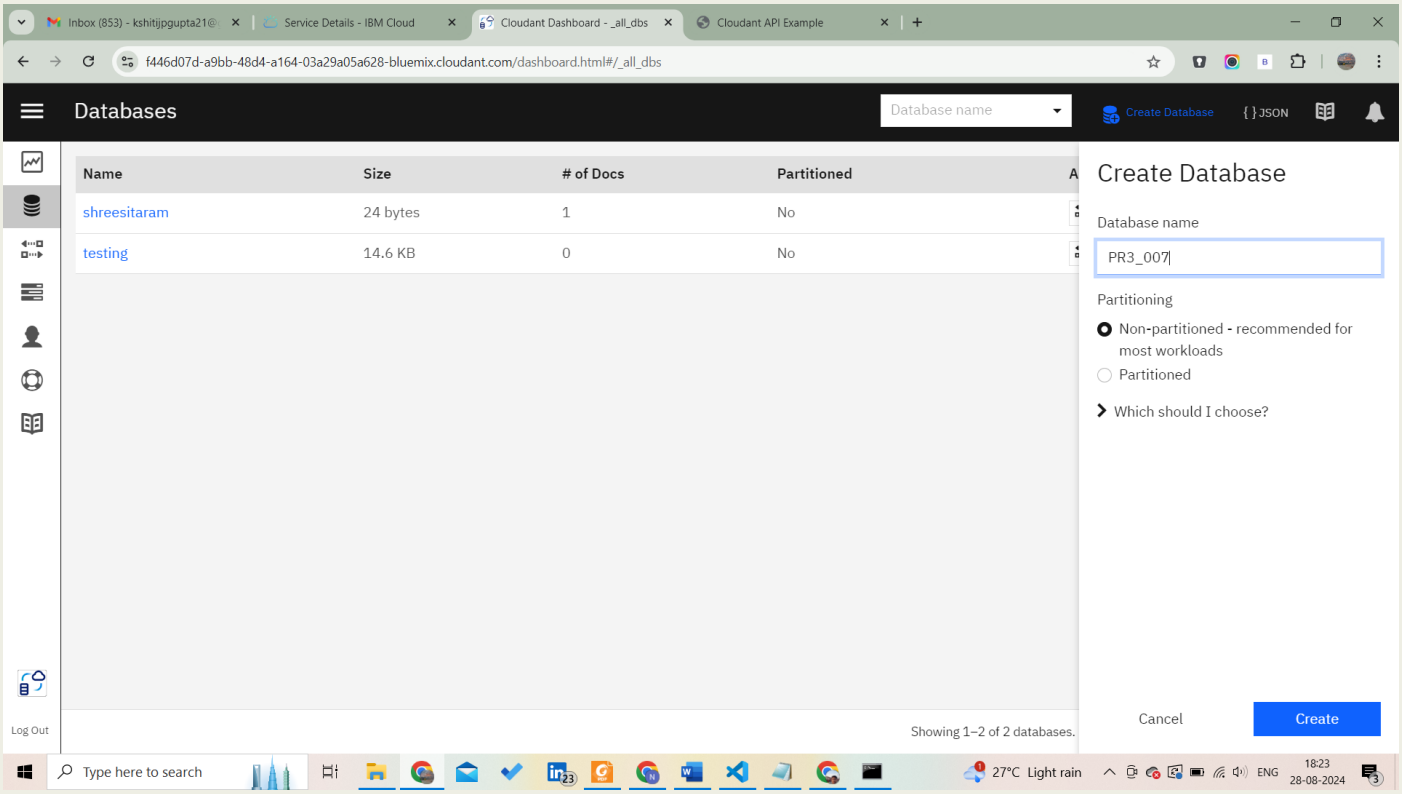




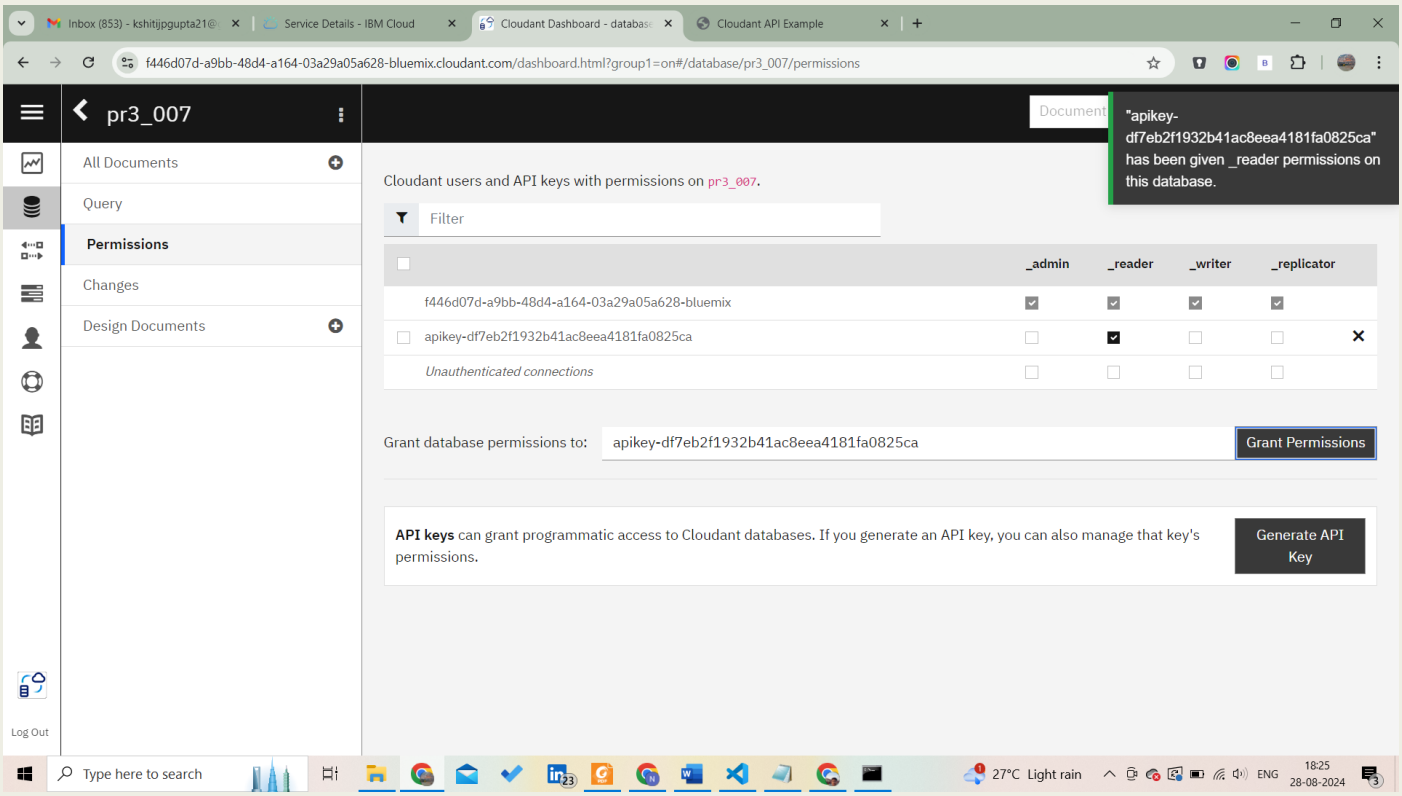

**Now we will try to replicate a database using the API key we generated.**

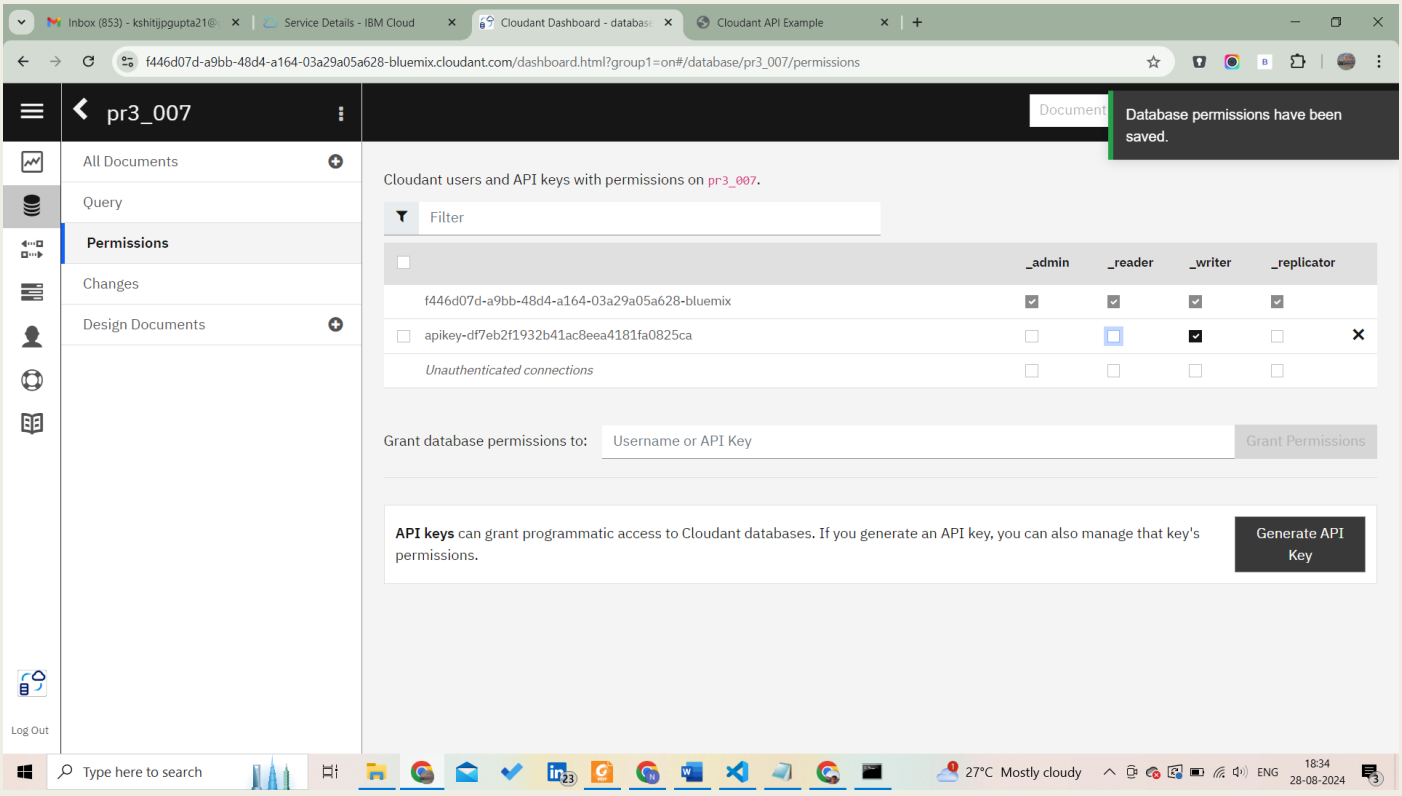
**For this we will first create a new database in which we will replicate the data. We need to have a database already created else this code will generate errors.**

**To create a new database in cloudant dashboard we will click on create database.**

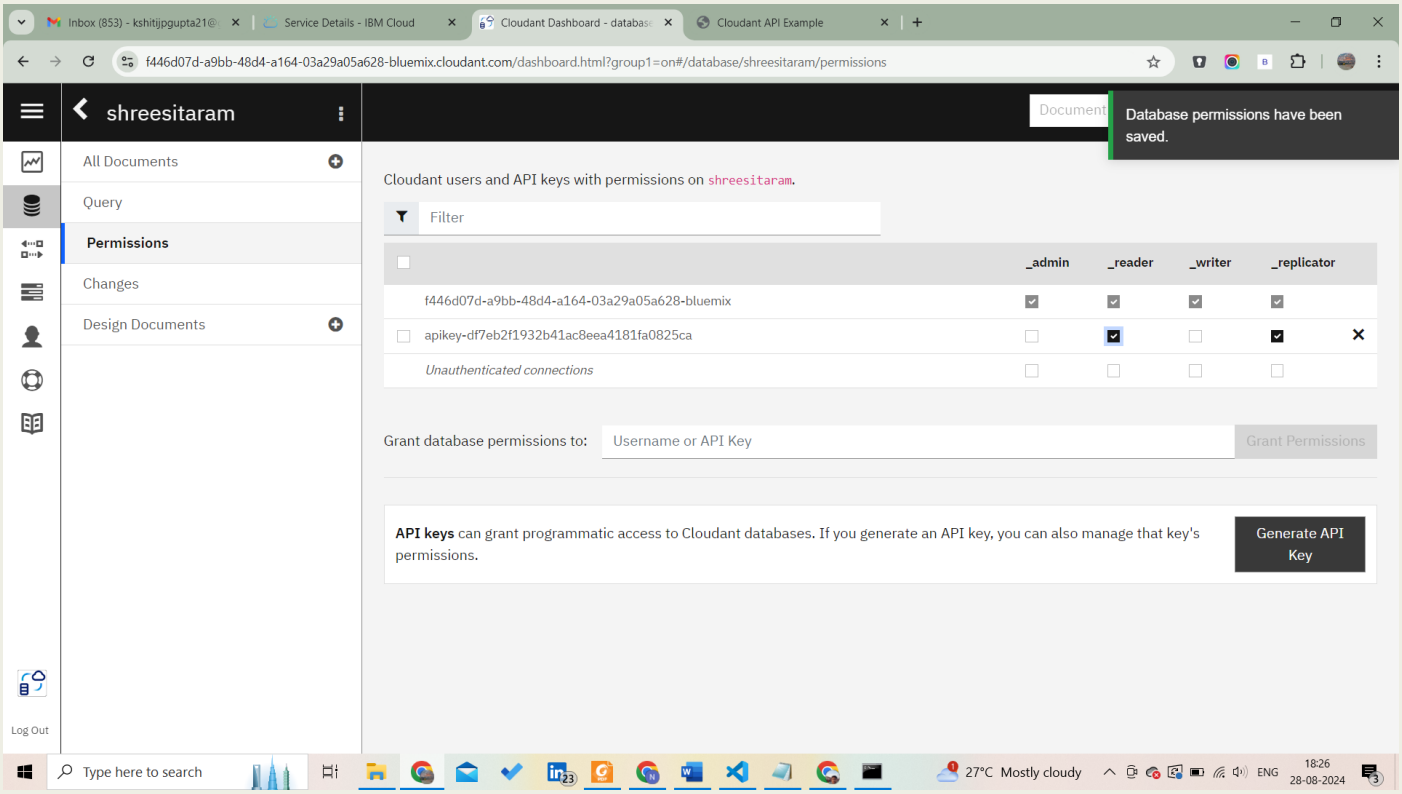


**Once this is created we copy the API key we previously created in this and grant access to write only.**




**Also we need to make sure that in the source database we also need to give the permission of the replicator. This permission is compulsory as without this it will not allow replication.**




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cloudant API Example</title>
</head>
<body>
    <h1>Cloudant API Example</h1>
    <button id="replicateData">Replicate Data</button>
    <pre id="result"></pre>

    <script>
        document.getElementById('replicateData').addEventListener('click', function ()
{
            const apiKey = 'apikey-df7eb2f1932b41ac8eea4181fa0825ca';
```

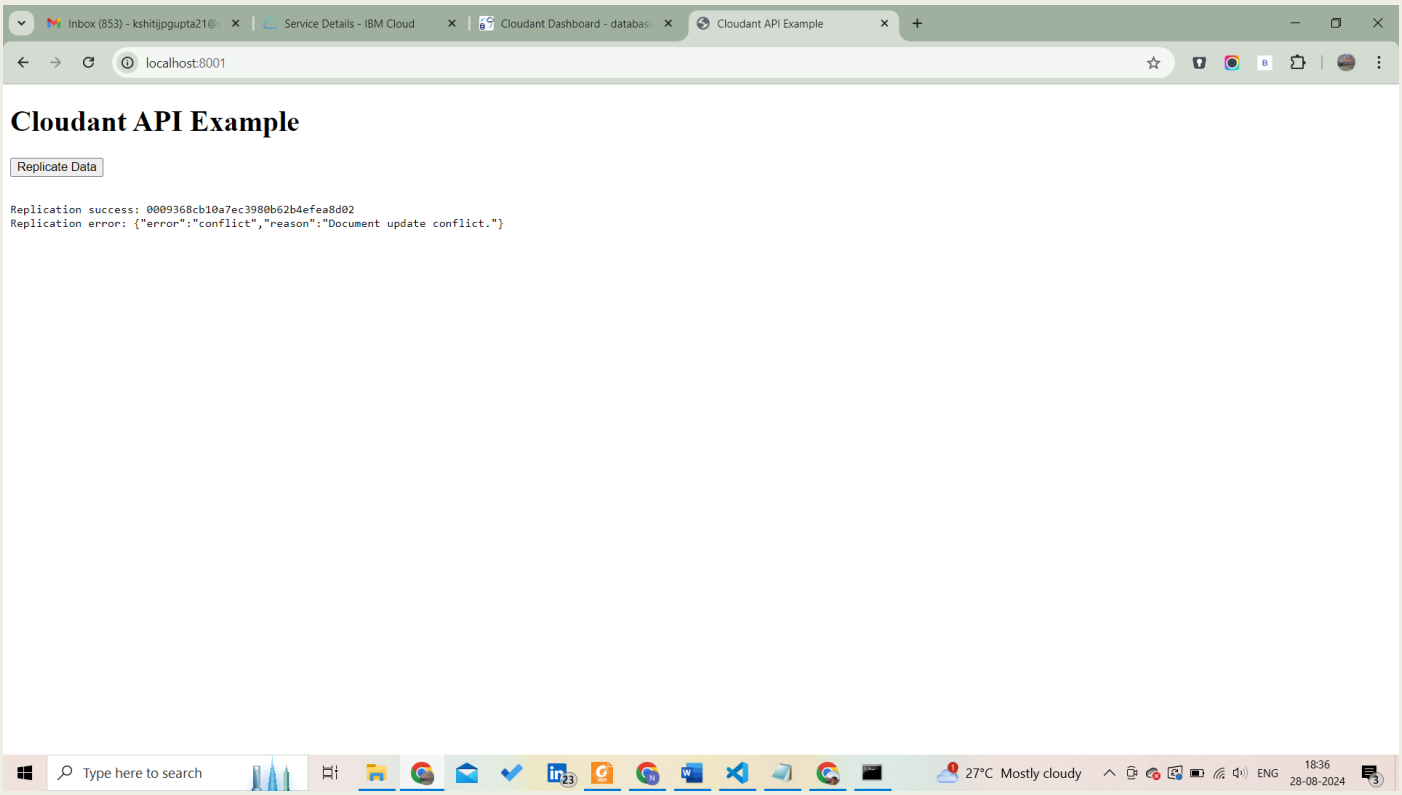
```
        const apiPassword = 'fe4d3890b2fdbe23db5782e38e8da0cb55fdff3f';
        const sourceUrl = 'https://f446d07d-a9bb-48d4-a164-03a29a05a628-bluemix.cloudantnosqldb.appdomain.cloud/shreesitaram/_all_docs?include_docs=true';
        const targetUrl = 'https://f446d07d-a9bb-48d4-a164-03a29a05a628-bluemix.cloudantnosqldb.appdomain.cloud/pr3_007';

        const headers = new Headers();
        headers.append('Authorization', 'Basic ' + btoa(apiKey + ':' + apiPassword));
        headers.append('Content-Type', 'application/json');

        fetch(sourceUrl, { method: 'GET', headers: headers })
            .then(response => response.json())
            .then(data => {
                const docs = data.rows.map(row => row.doc);
                docs.forEach(doc => {
                    replicateDocument(doc, targetUrl, headers);
                });
            })
            .catch(error => {
                document.getElementById('result').textContent = 'Replication error: ' + error;
            });
    });

    function replicateDocument(doc, targetUrl, headers) {
        const docUrl = `${targetUrl}/${doc._id}`;

        fetch(docUrl, { method: 'GET', headers: headers })
```

```
            .then(response => {
                if (response.status === 404) {
                    // Document does not exist, create it
                    return fetch(targetUrl, {
                        method: 'POST',
                        headers: headers,
                        body: JSON.stringify(doc)
                    });
                } else {
                    return response.json().then(existingDoc => {
                        // Document exists, update it
                        doc._rev = existingDoc._rev;
                        return fetch(docUrl, {
                            method: 'PUT',
                            headers: headers,
                            body: JSON.stringify(doc)
                        });
                    });
                }
            })
            .then(response => response.json())
            .then(data => {
                if (data.ok) {
                    document.getElementById('result').textContent += `\nReplication success: ${doc._id}`;
                } else {
                    document.getElementById('result').textContent += `\nReplication error: ${JSON.stringify(data)}`;
```

```
            }
        })
        .catch(error => {
            document.getElementById('result').textContent += `\nReplication error: ${error.message}`;
        });
    }
  </script>
</body>
</html>
```

**Here we are giving the source url and target url including the hostname along with the database name, also the generated API key and password are sent encoded with request.**

**Now if we run the code and the permissions and password are correct then we get success.**



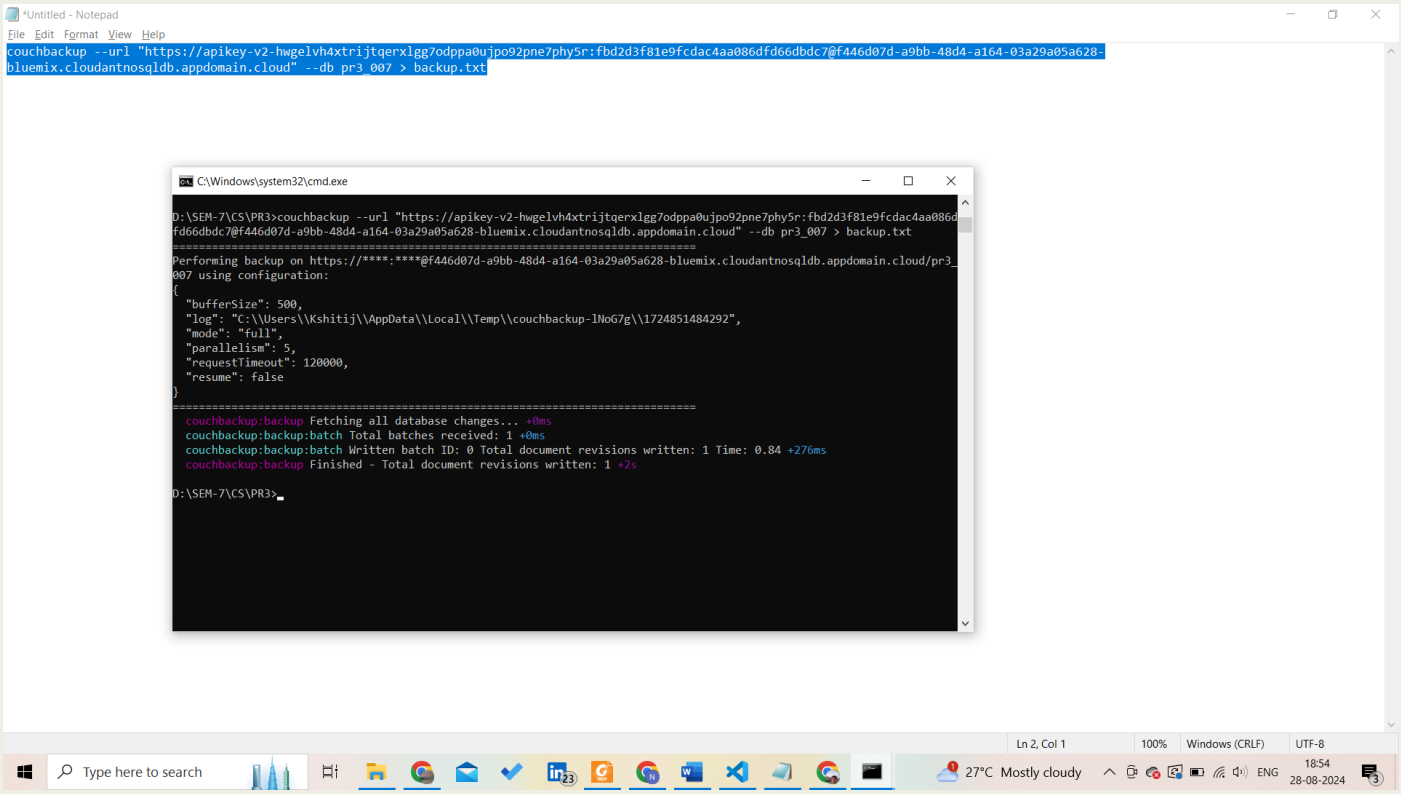

**In the above case I gave writer access to the target database hence we were able to replicate the data , but if I don't give writer permission then it will**

**4. Now we will see Compaction, which is a process that helps manage and optimize the storage of documents within the database. When documents are deleted or updated, the previous versions of these documents are not immediately removed. Instead, they are marked for deletion and removed later during the compaction process.**

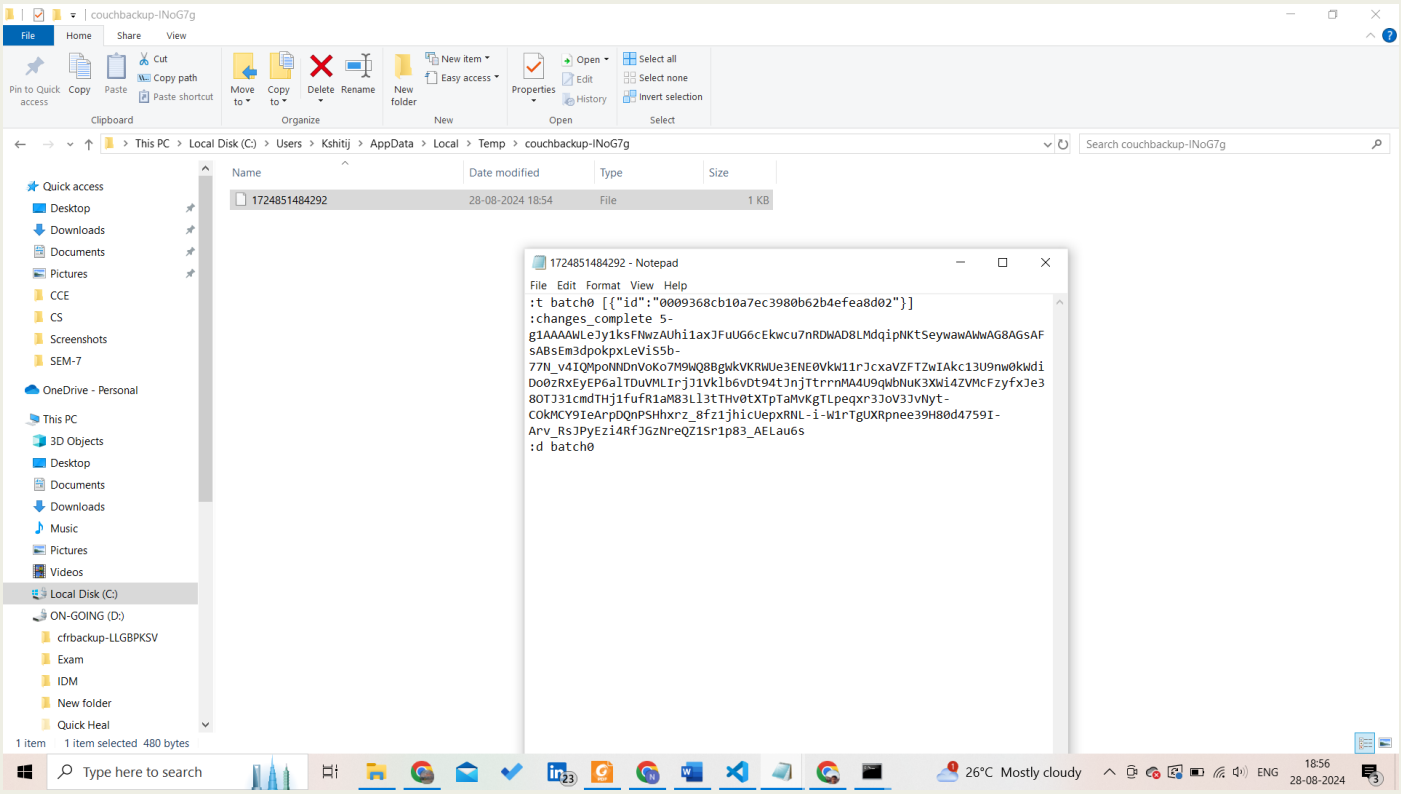
**Now this is automatically done by IBM itself; we can not do it ourselves.**

**The next thing we are doing is to take backup in text file format. We are using a package called couchbackup. So first we will install this package and then in the command prompt.**

**First I am going to take backup of a database and store it in my machine.**

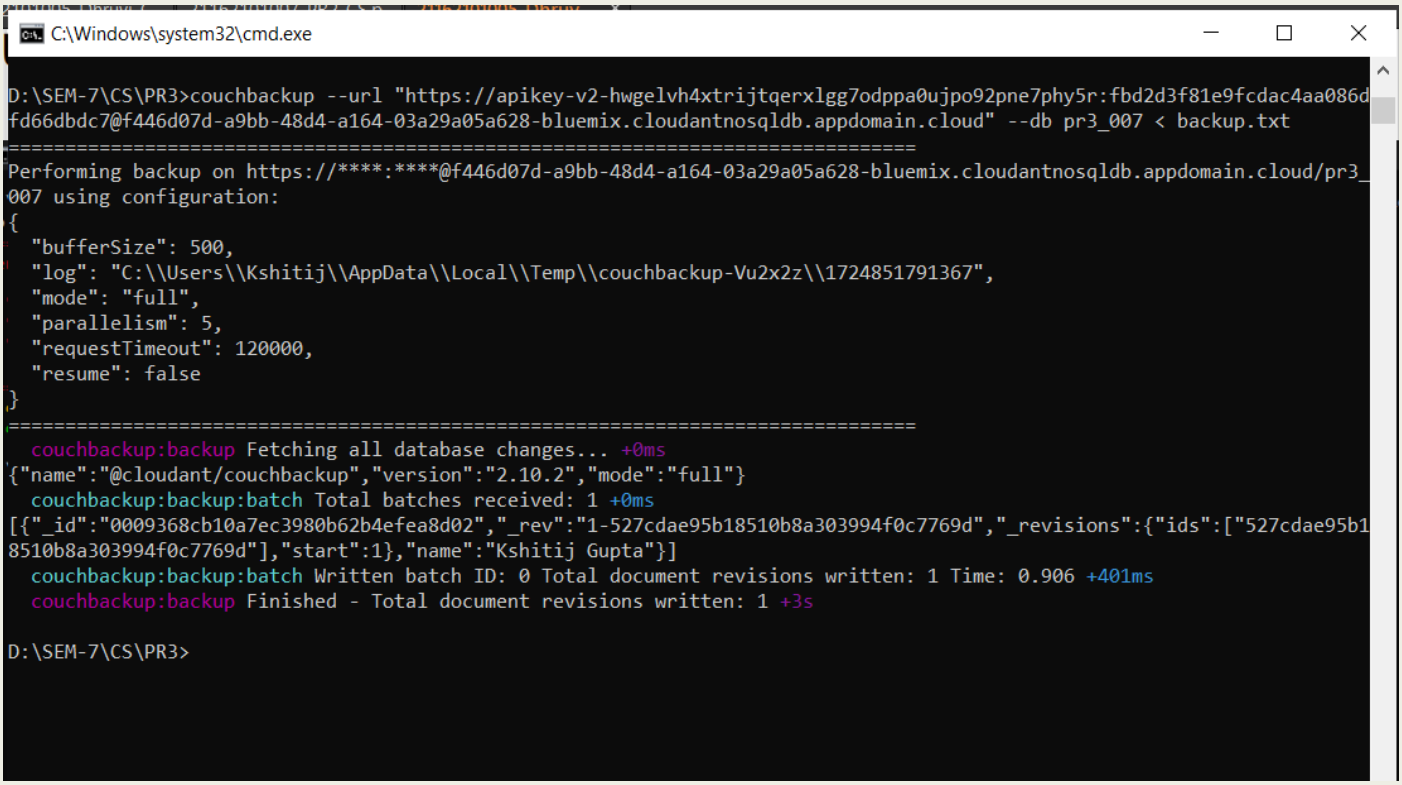


**Now if I check I will have a text file named backup.**

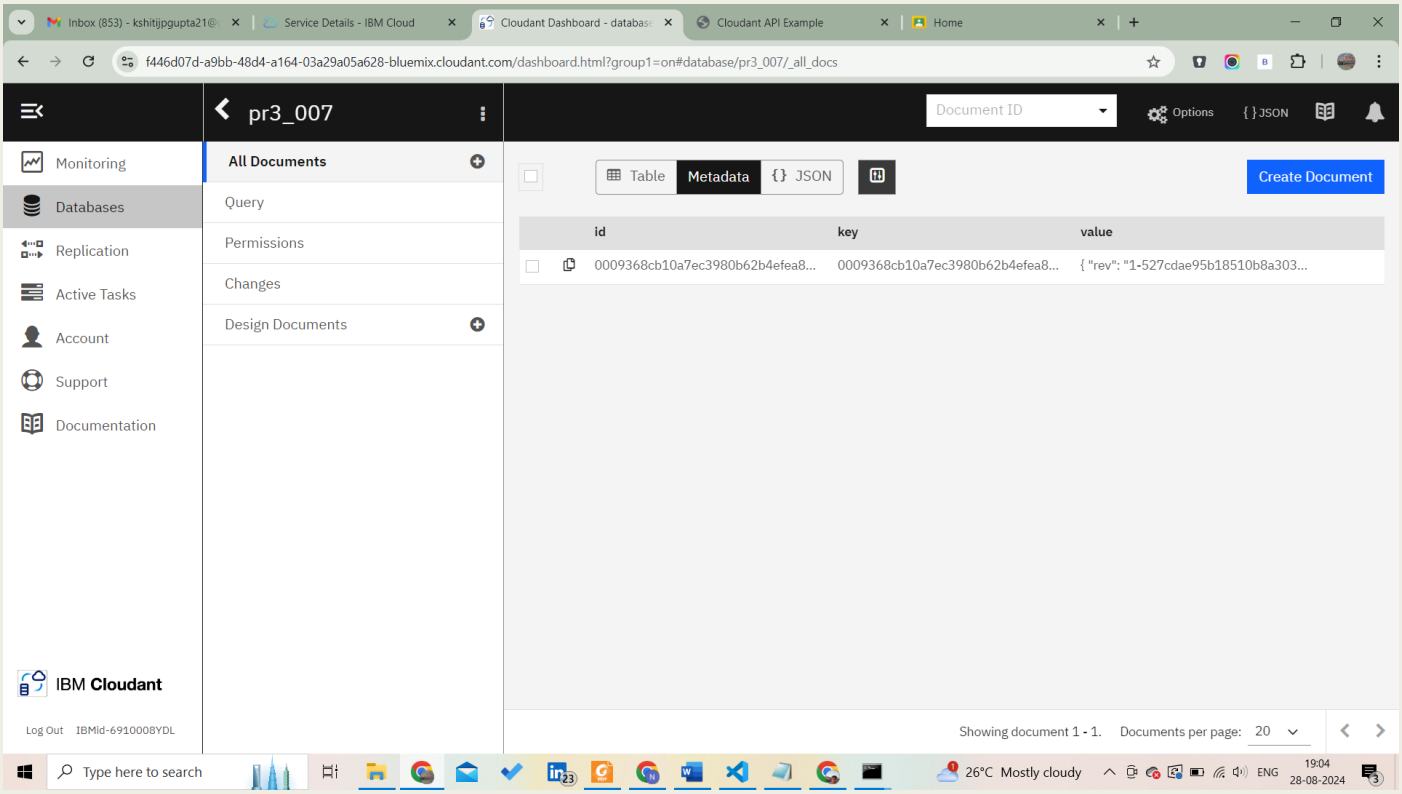



**Now from this text file I want to add this data in a database in cloudant.**

**couchbackup --url "https://apikey-v2-hwgelvh4xtrijtqerxlgg7odppa0ujpo92pne7phy5r:fbd2d3f81e9fcdac4aa086dfd66dbdc7@f446d07d-a9bb-48d4-a164-03a29a05a628-bluemix.cloudantnosqldb.appdomain.cloud" --db pr3_007 < backup.txt**



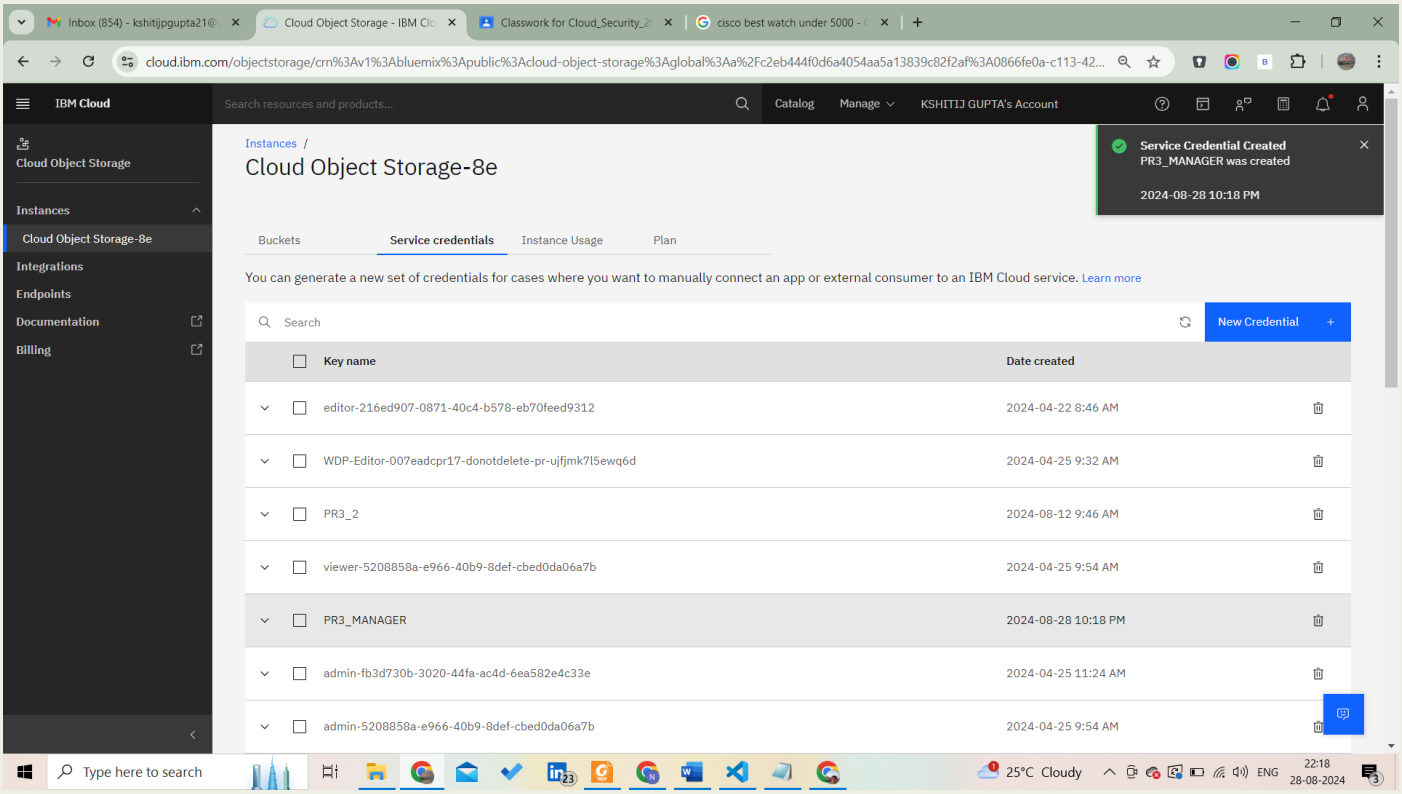
**Now if we check the database we will have the same data here.**



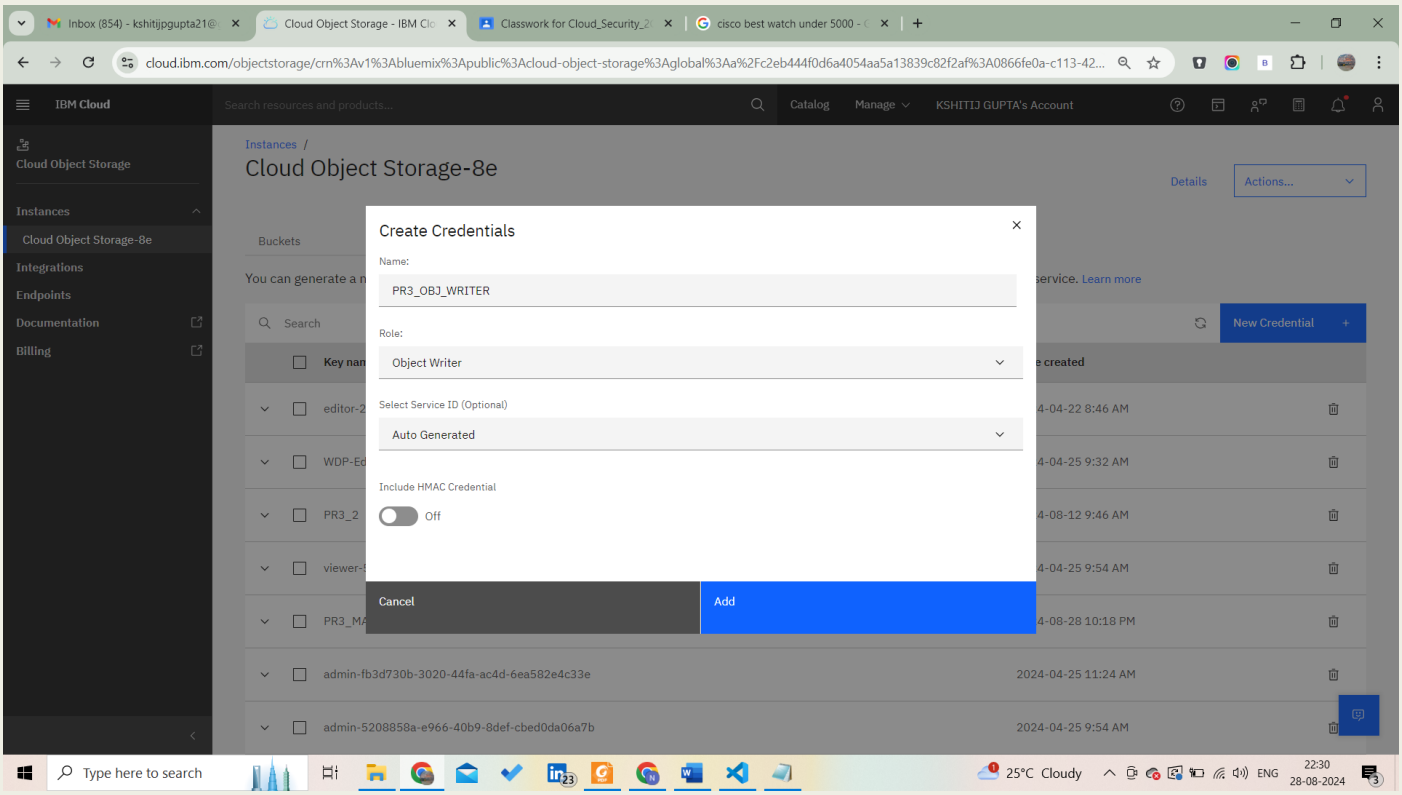
## Object storage:

**Now in the instance of cloud object storage we will create 3 new credentials: Manager, Object writer and Reader. In Service Credentials click on New Credential and here we can give name and select the role.**

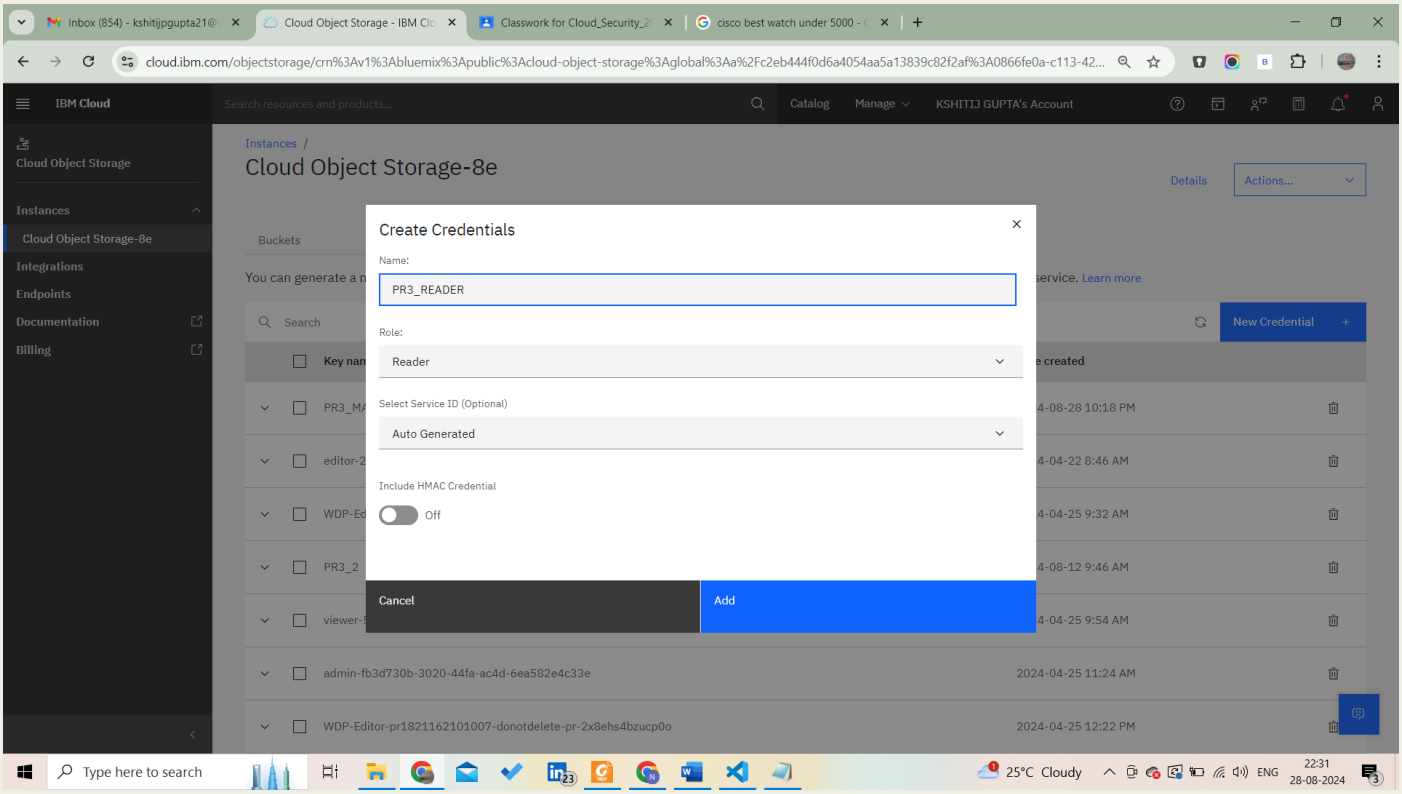
## MANAGER:



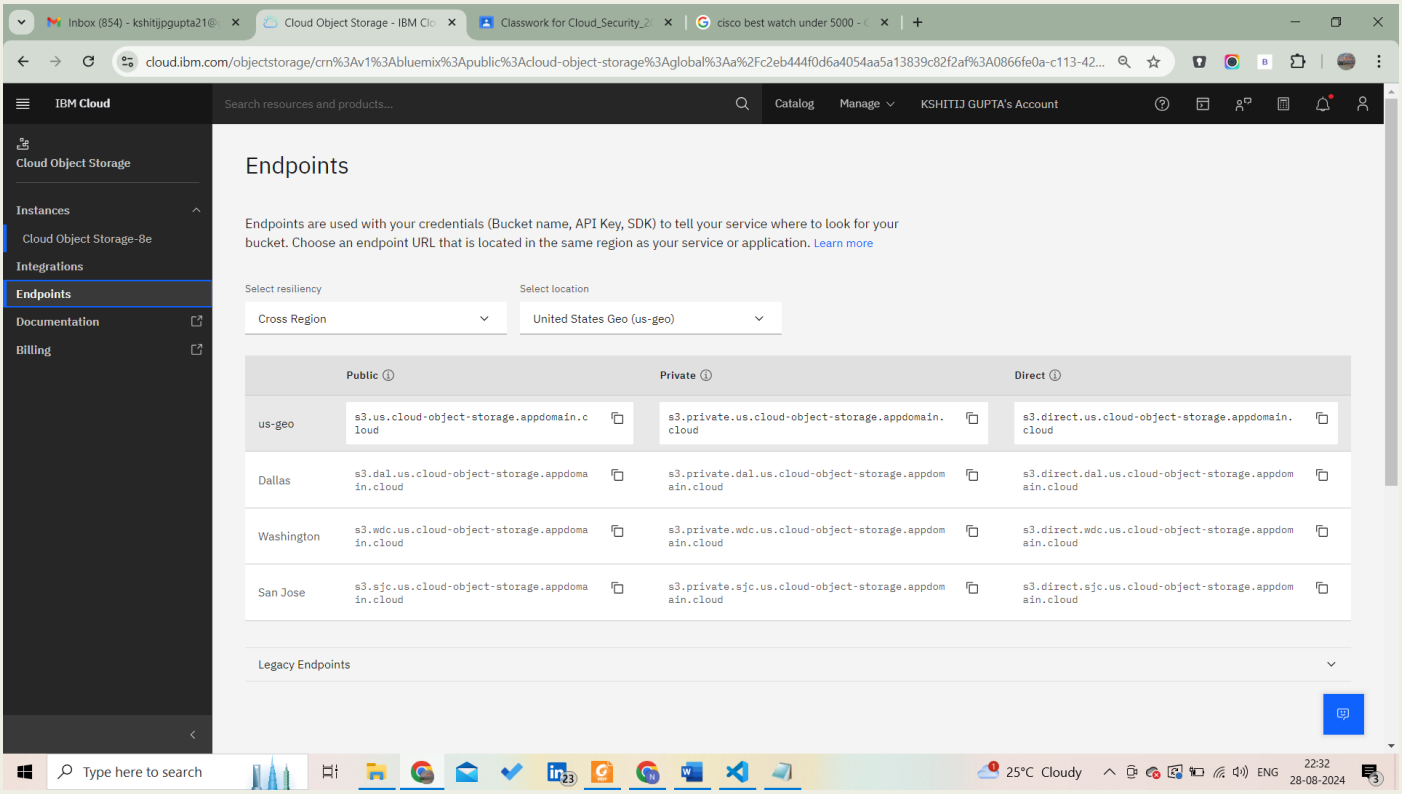

## OBJECT-WRITER



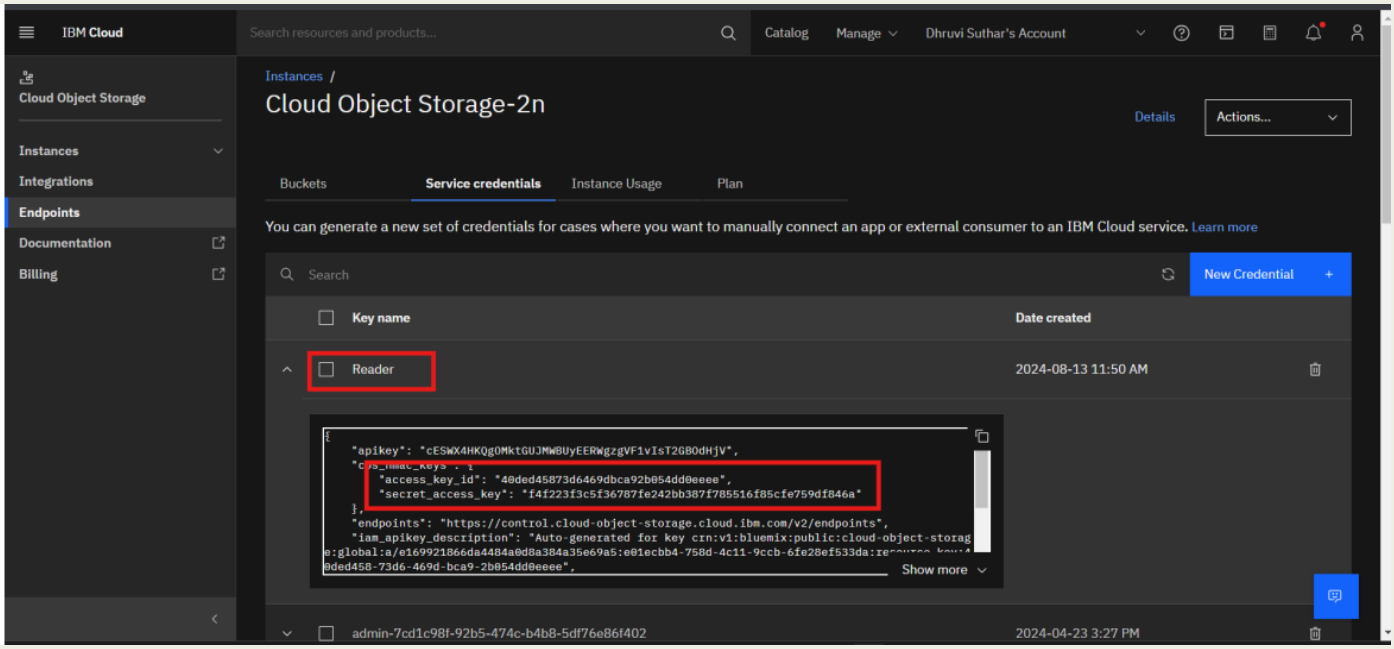
## Reader:



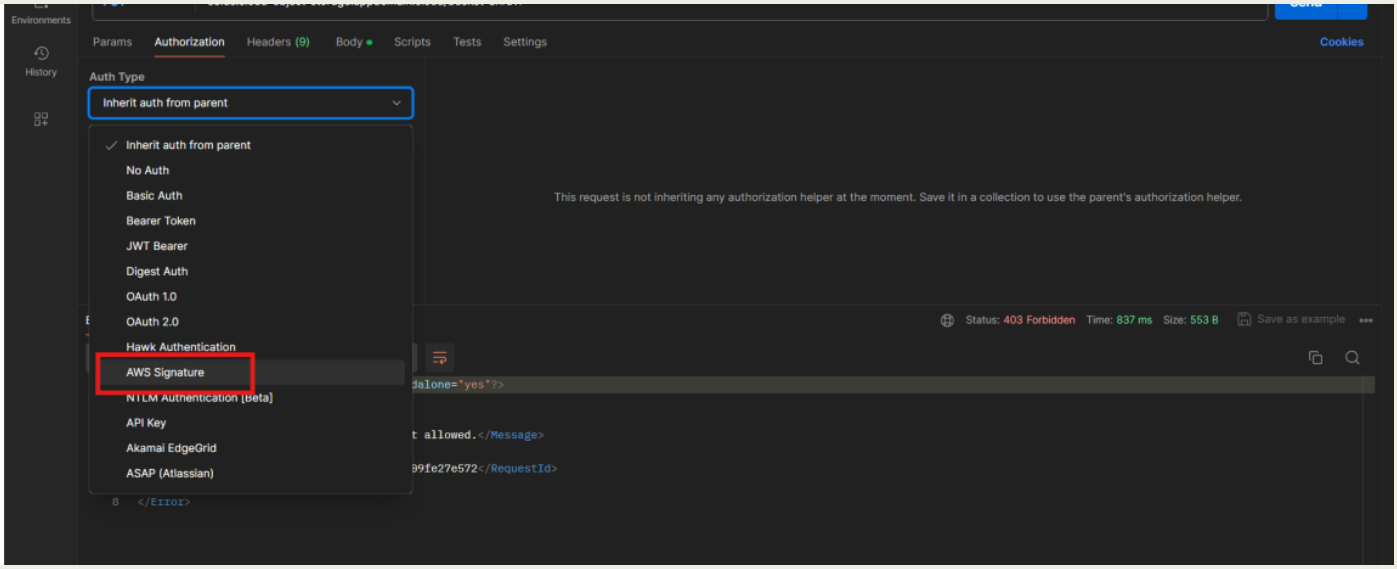

**Now in the Endpoints we can copy the public URL and now go to postman and paste it without any Authorization.**



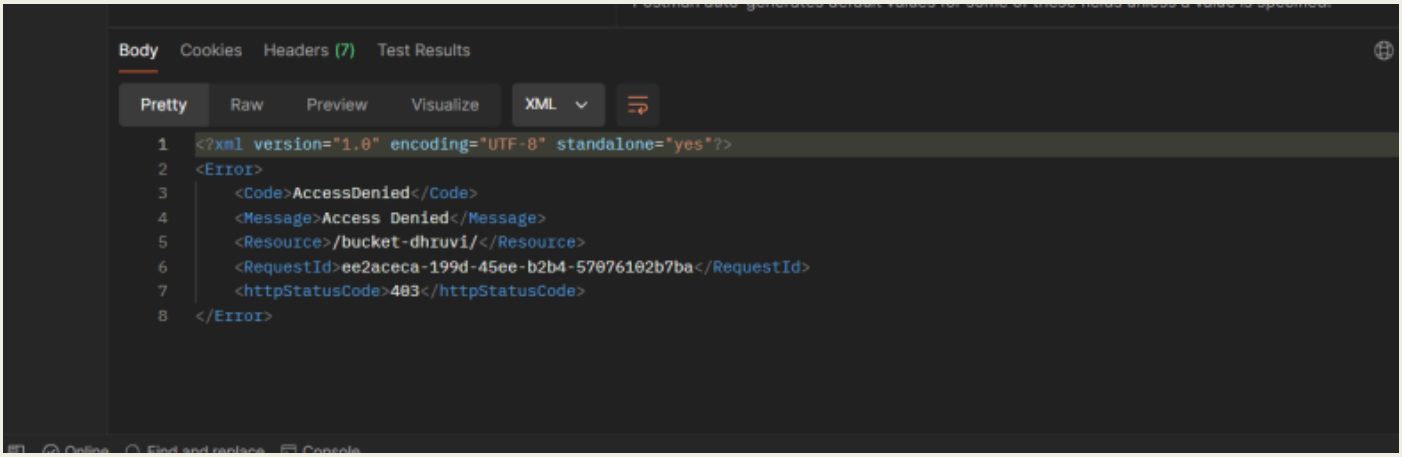

**Now here we tried to create a bucket but as we did not give any authorization hence we could not create the bucket and Now we can try the access key an ID of Reader Role as authorization.**



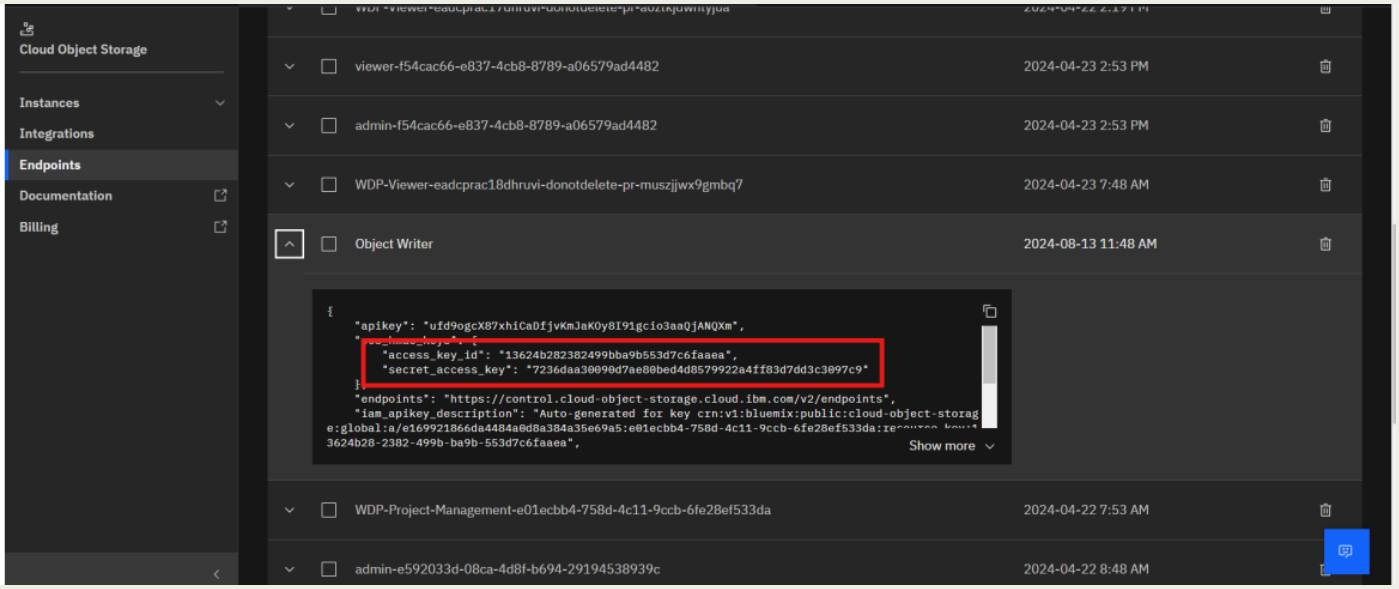

**In authorization we selected AWS Signature**

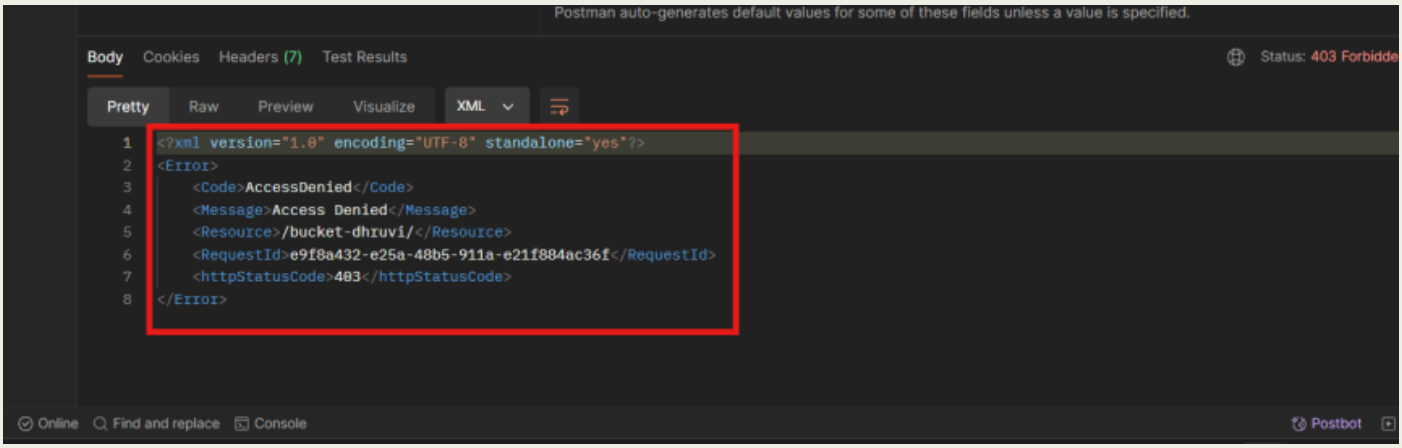



**Now in Reader we only have access to read hence we can not create a bucket**

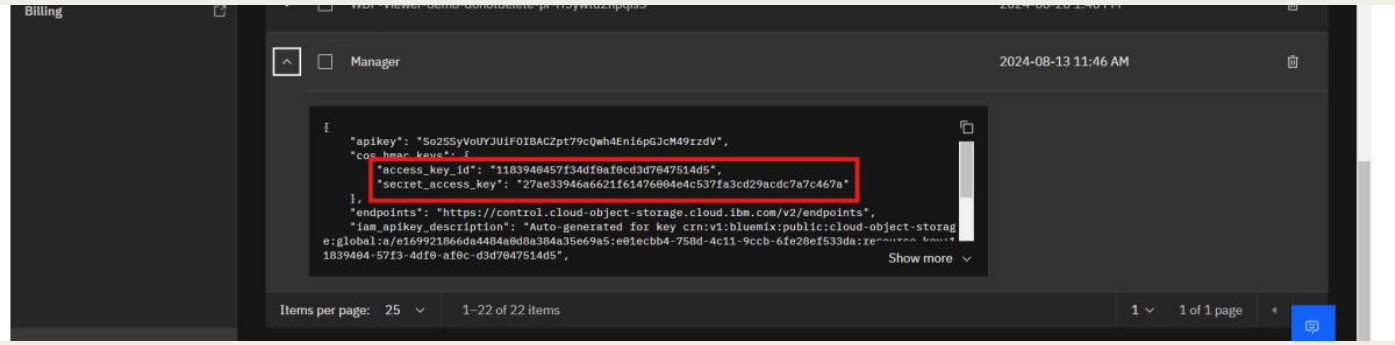



**Similarly we can take the access keys from the Object writer and try to create a bucket. Here also we will be denied the access.**

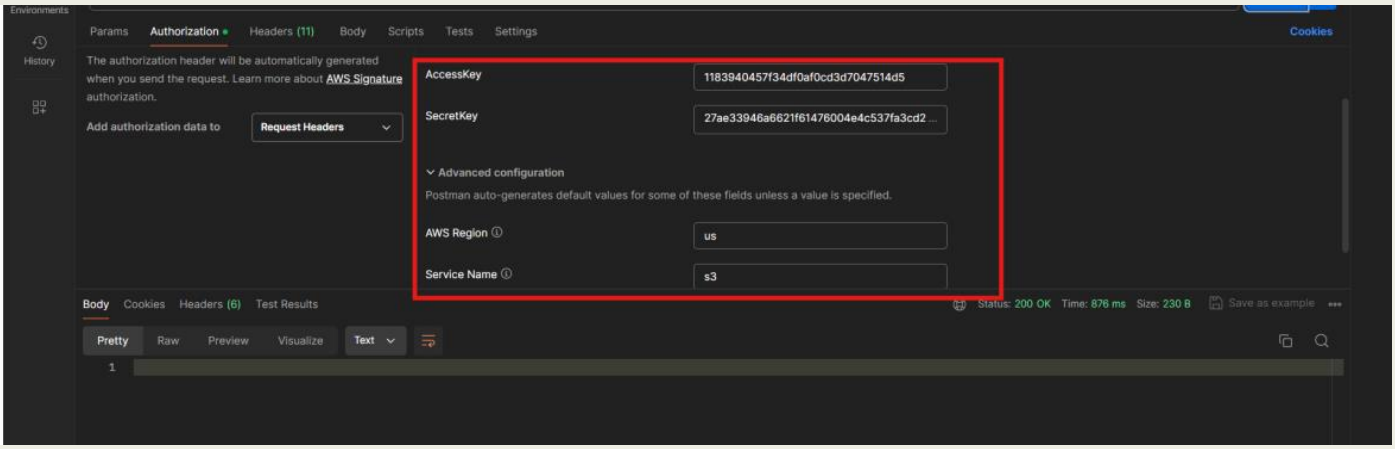


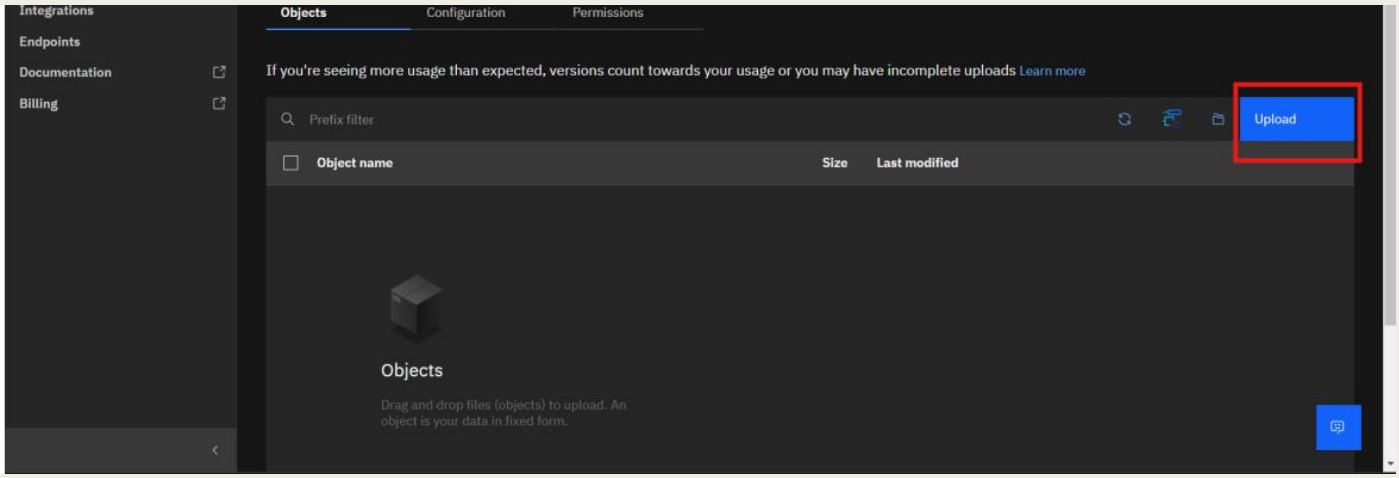

## Now we will try for the Manager role in postman to create a bucket

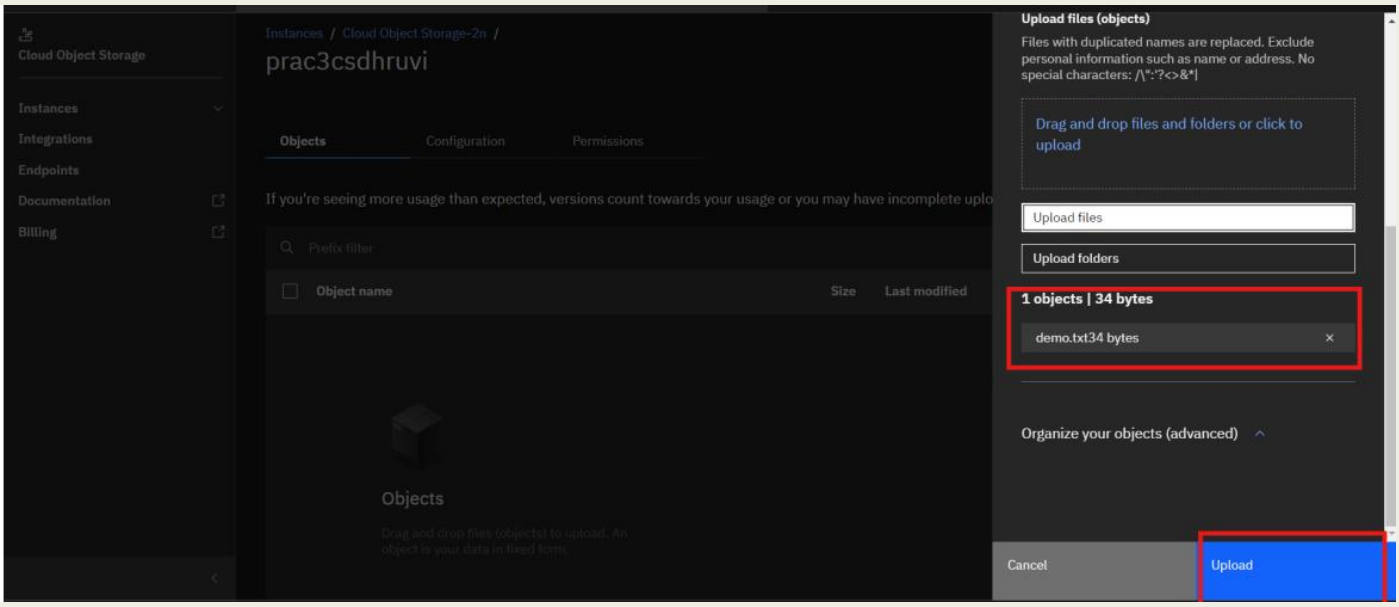



## Here as we can see that the bucket is created

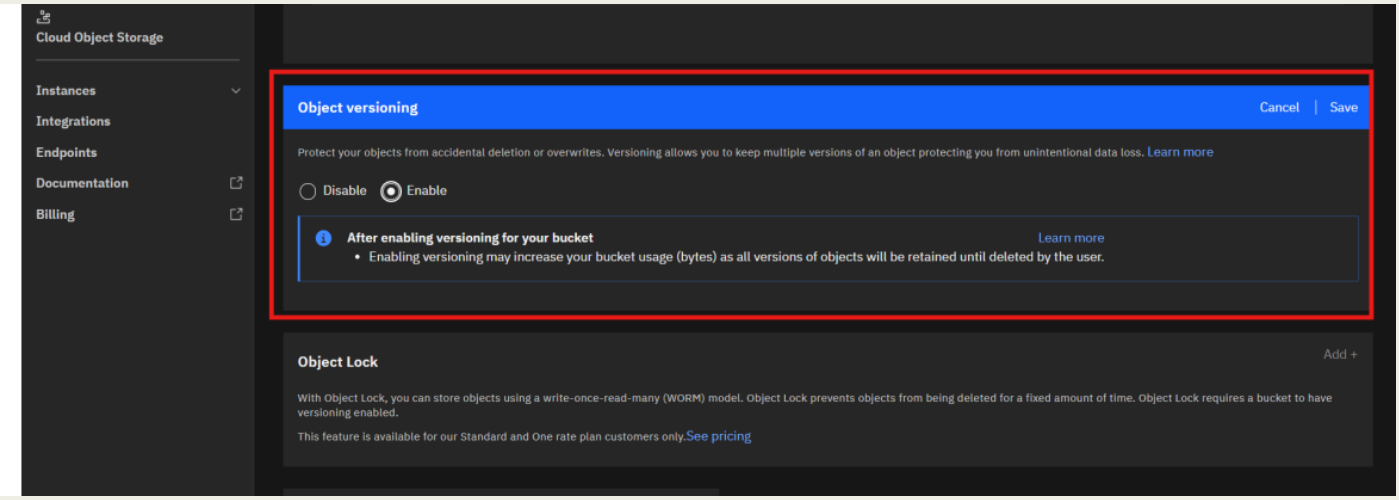


**Now we will check the versioning in the bucket we just created. We will upload a file.**

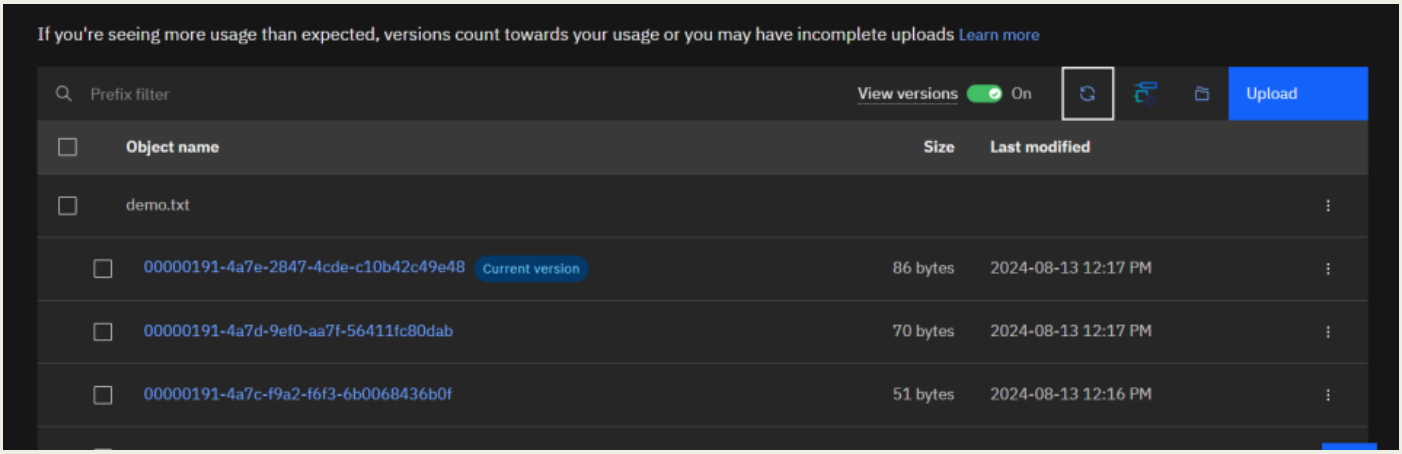


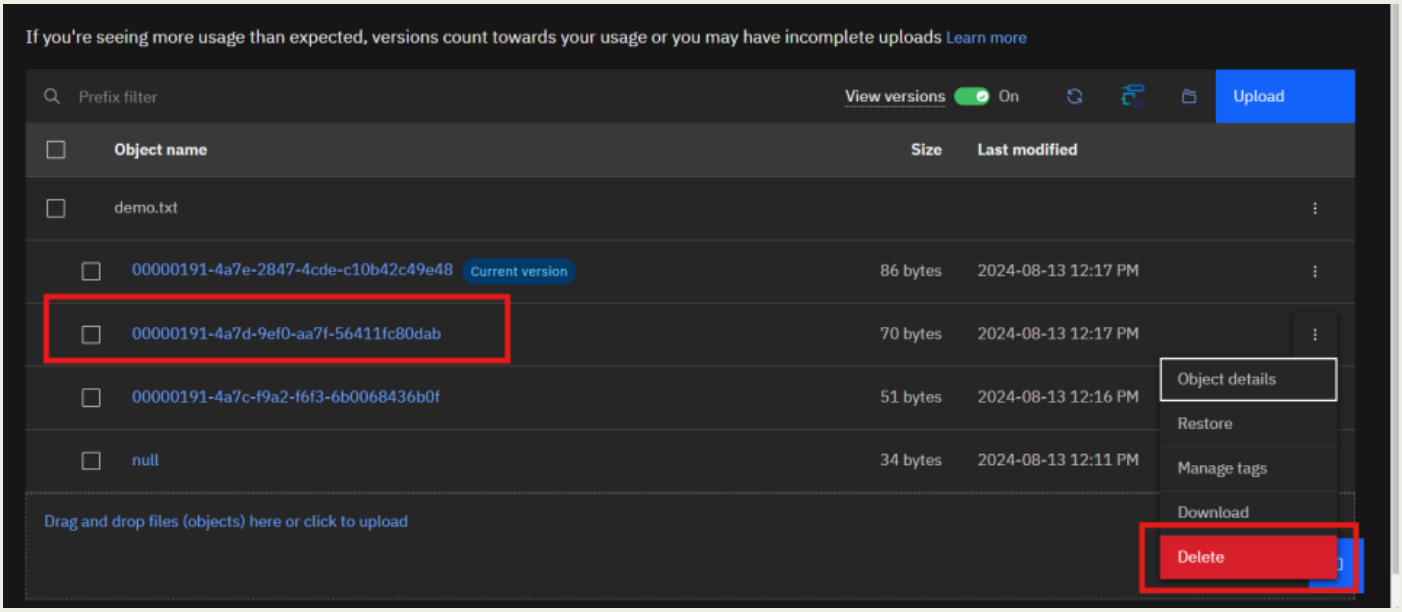

**Now enable Object Versioning.**

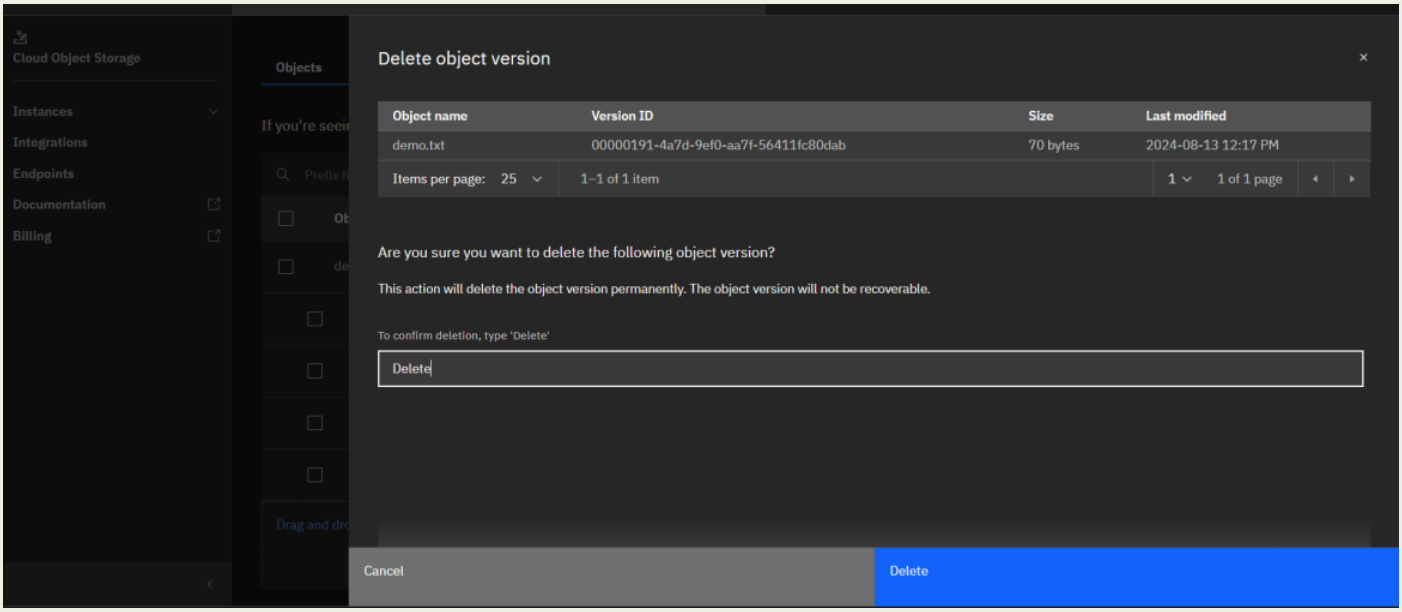



**Now we can upload multiple versions of a file and once we enable the View Versions, now we can see all the versions here.**

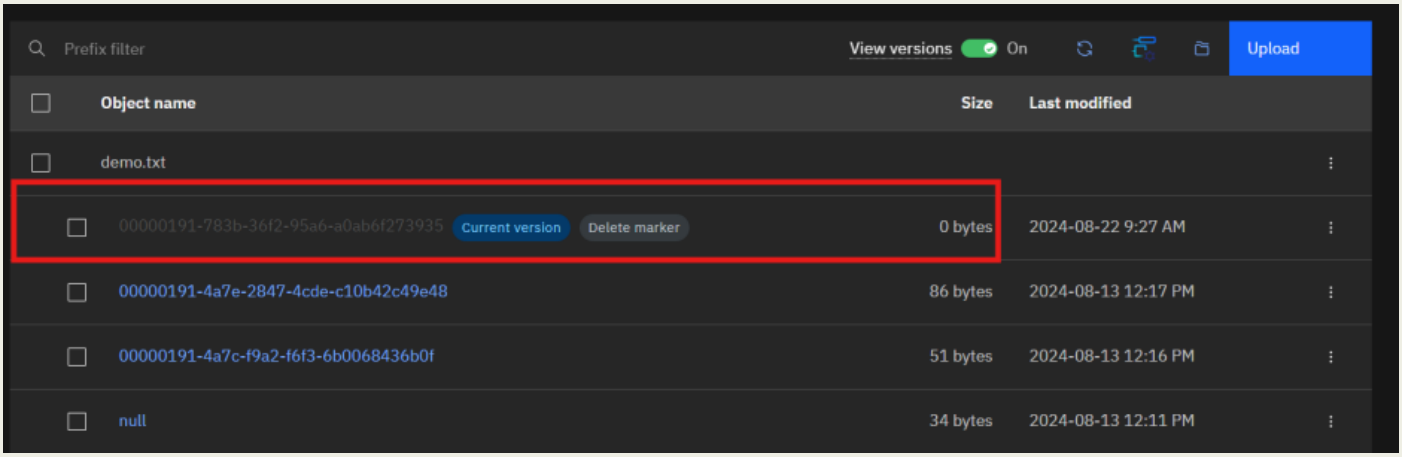



**Now if we delete any older versions we can see that it is deleted and removed from the list too.**

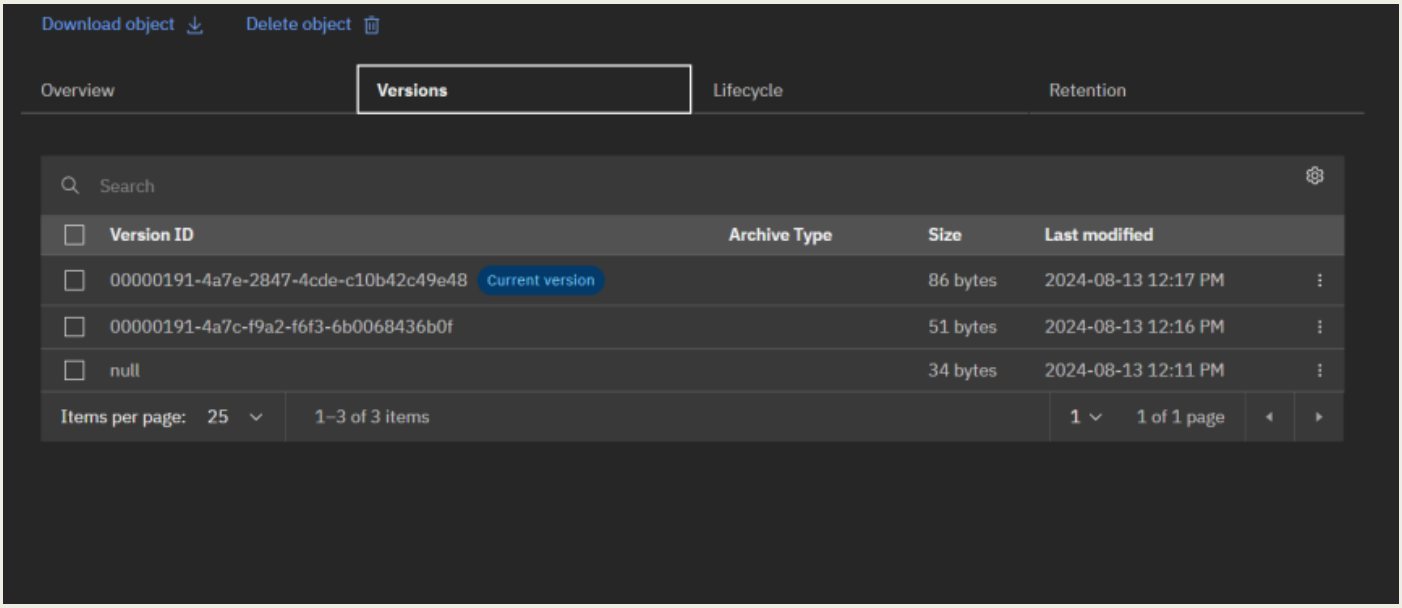


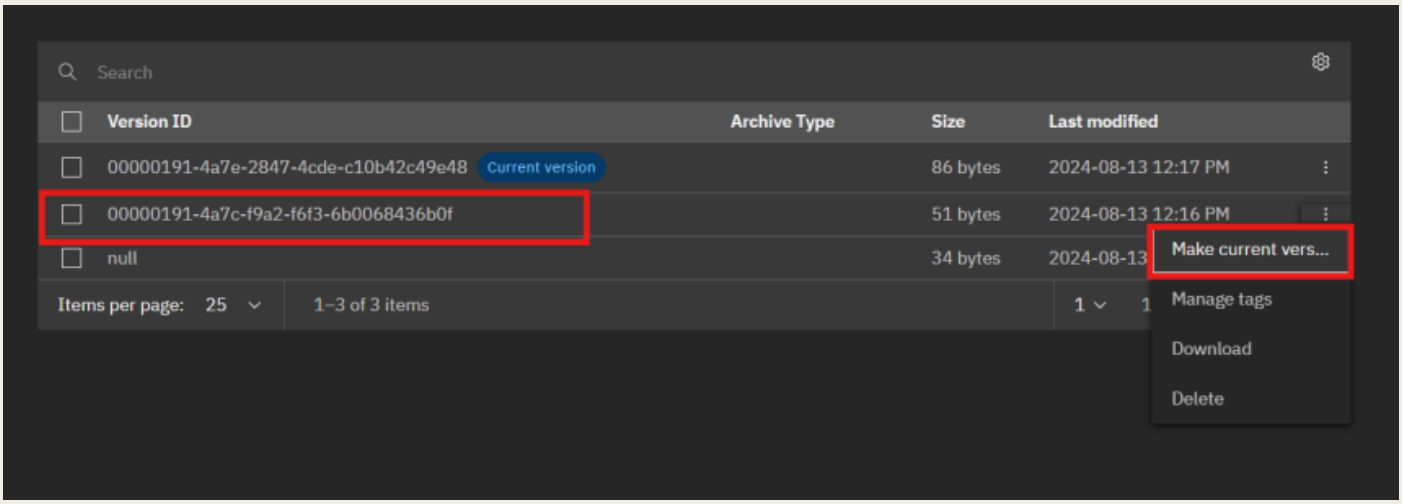

**But now if we try to delete the current version it is deleted but not removed instead delete market is there and we can also restore it**



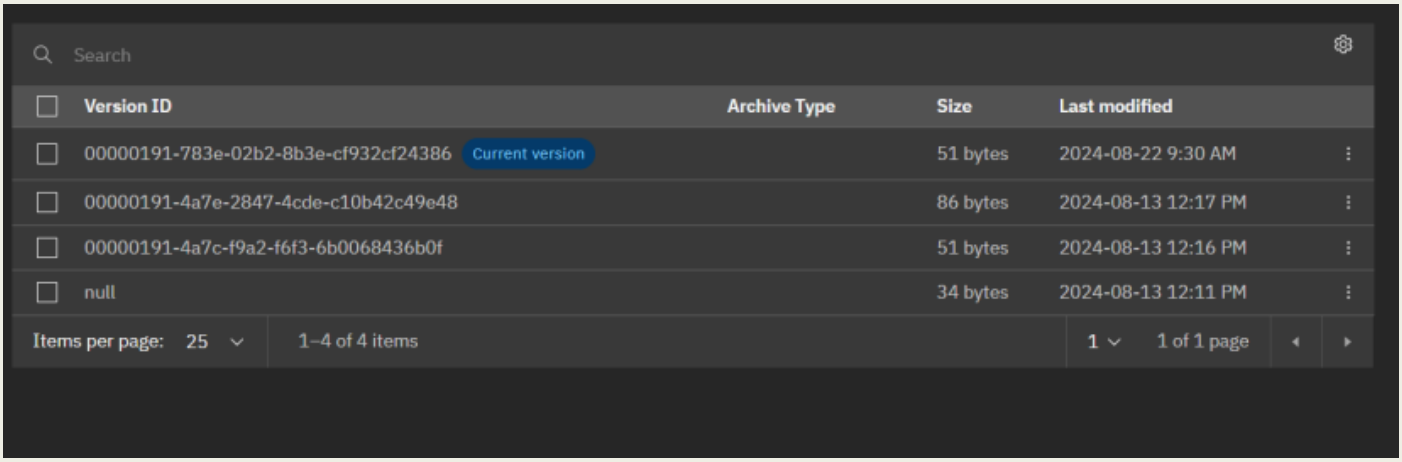

**Now if I want to change the current version of the object we can go to the Versions**




**Here select the version you want to make current version.**




**And as we can see the current version is changed.**


Search
Version ID
Archive Type
Size
Last modified
00000191-783e-02b2-8b3e-cf932cf24386
Current version
51 bytes
2024-08-22 9:30 AM
00000191-4a7e-2847-4cde-c10b42c49e48
86 bytes
2024-08-13 12:17 PM
00000191-4a7c-f9a2-f6f3-6b0068436b0f
51 bytes
2024-08-13 12:16 PM
null
34 bytes
2024-08-13 12:11 PM
Items per page: 25
1–4 of 4 items
1
1 of 1 page