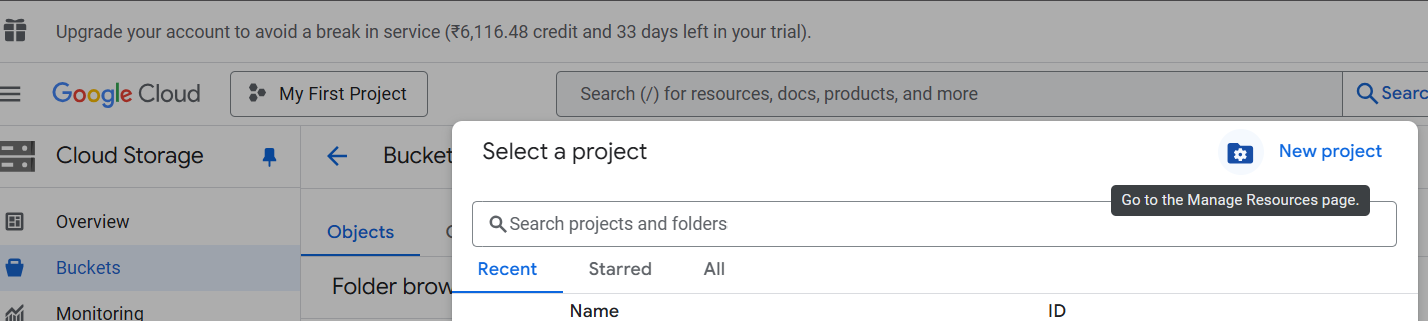
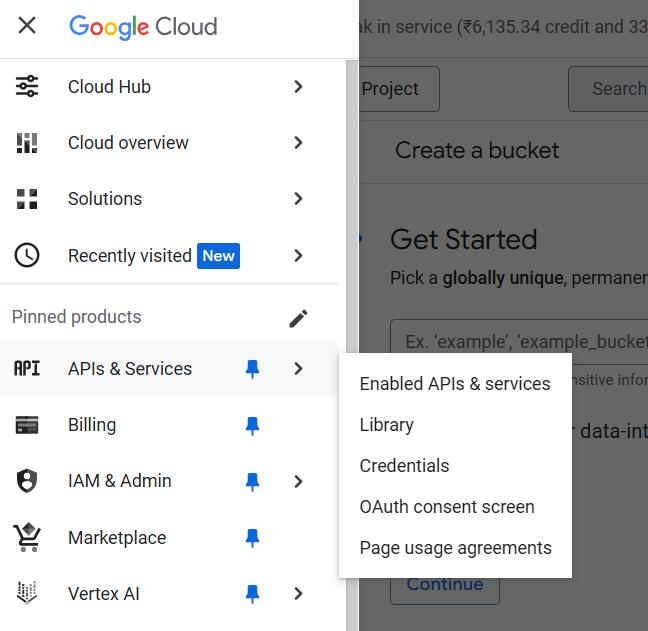
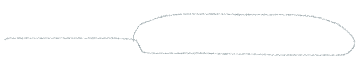


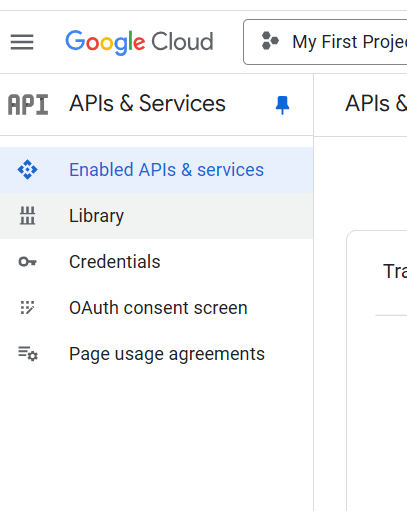
Create project first



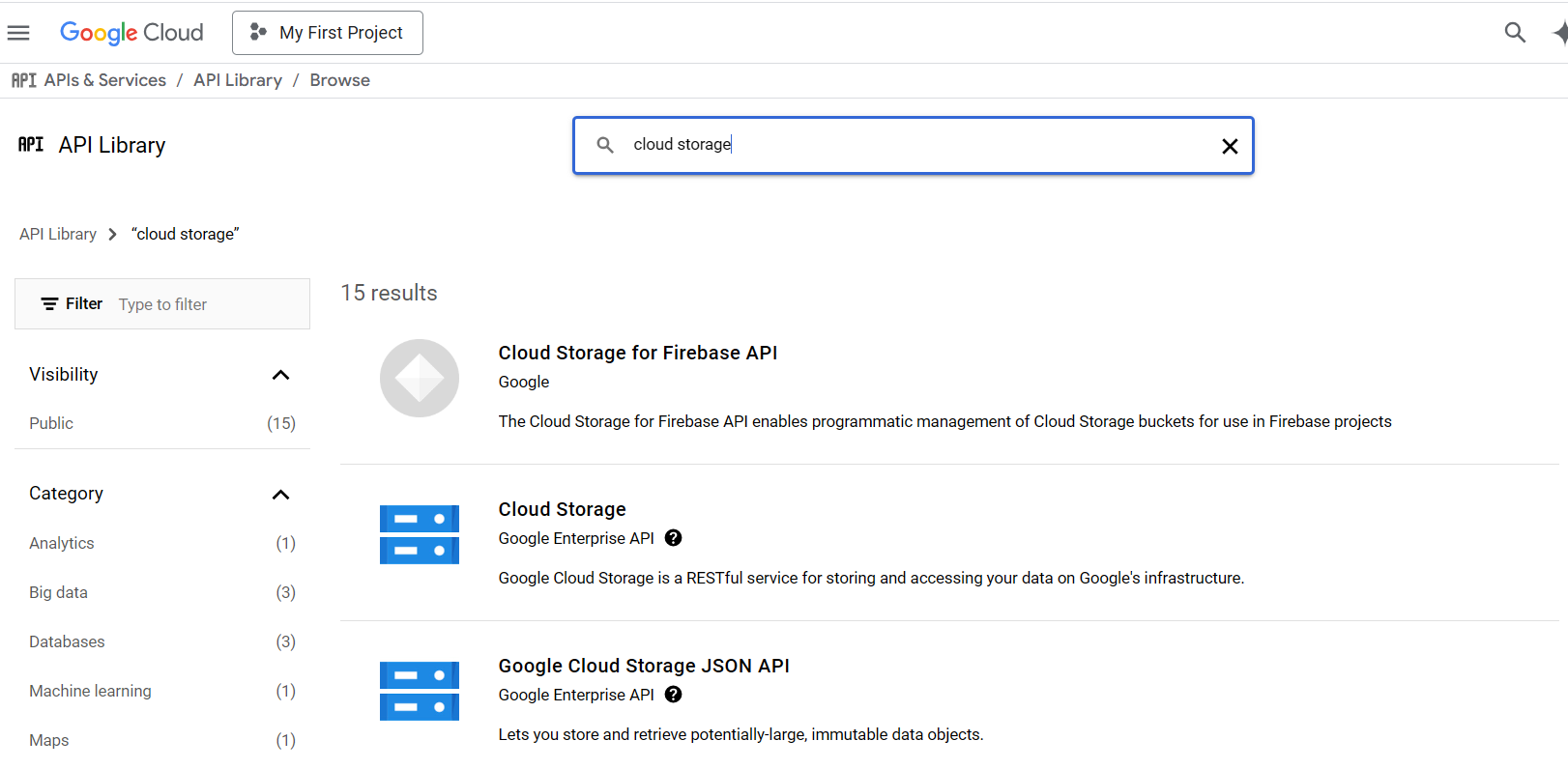


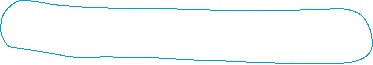


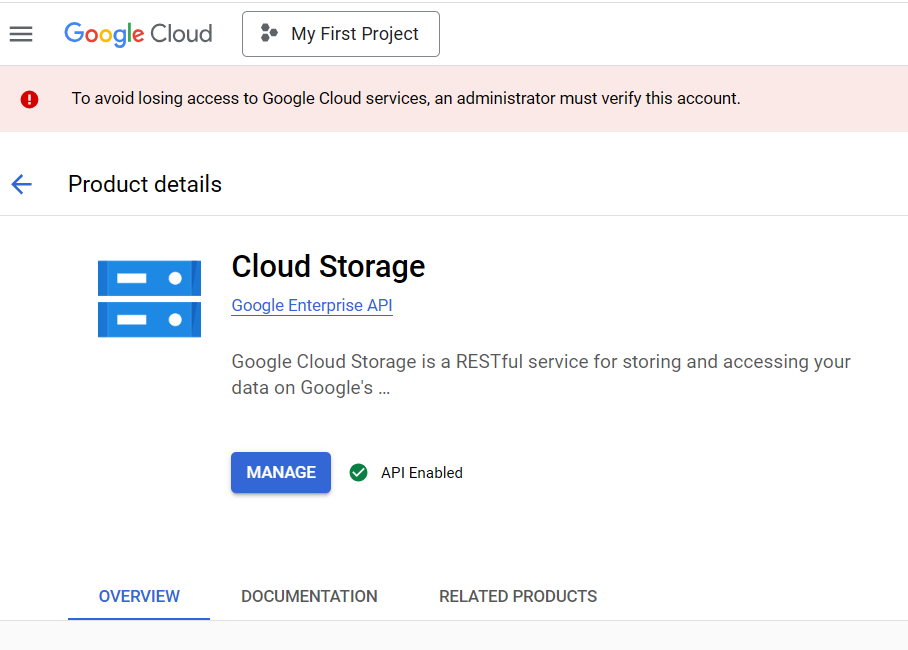






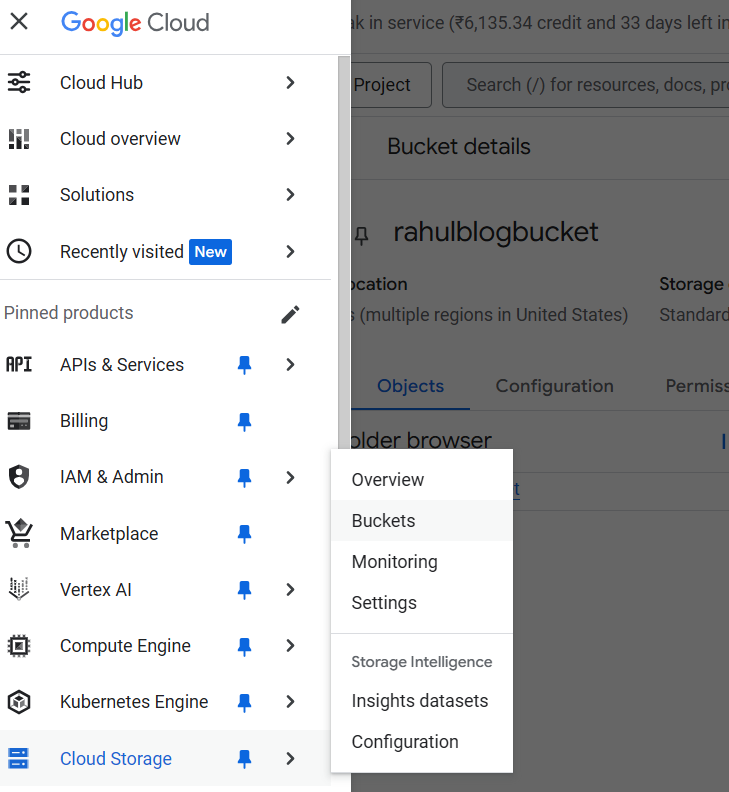


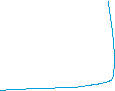
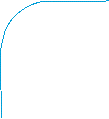




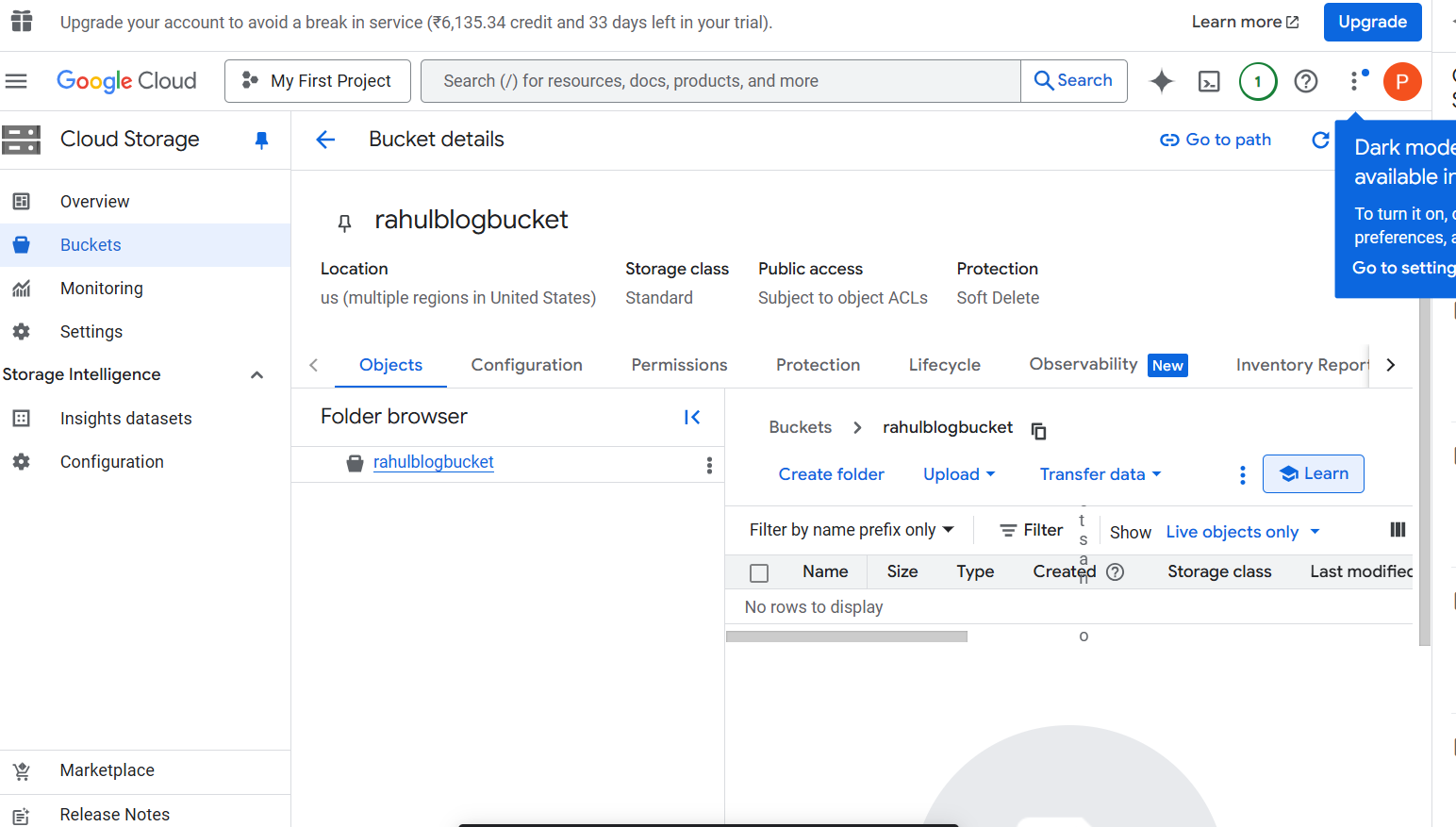


Create a storage bucket



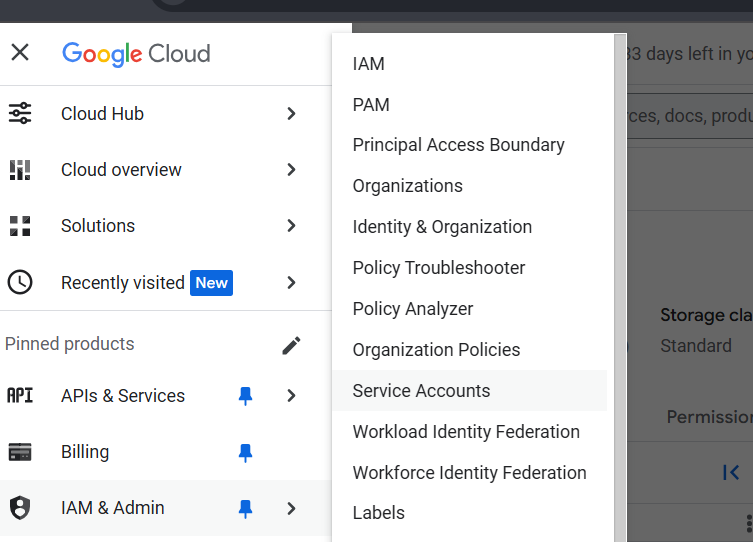


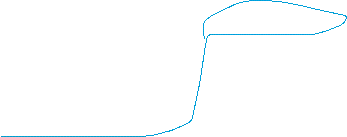
Now set IAM

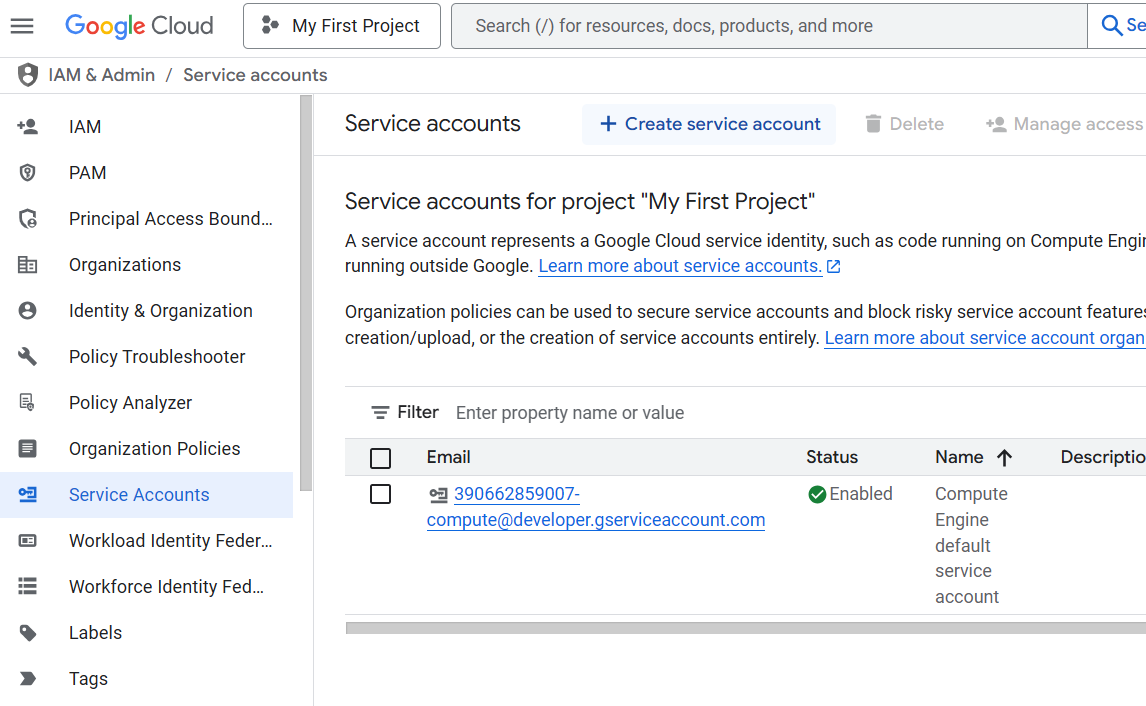




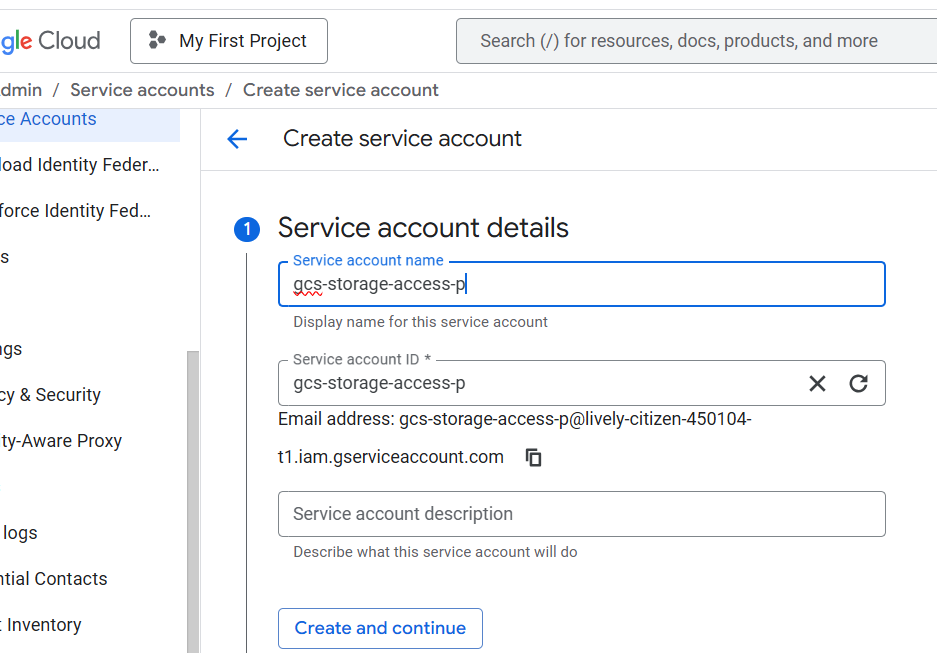
Now setup IAM-service accounts





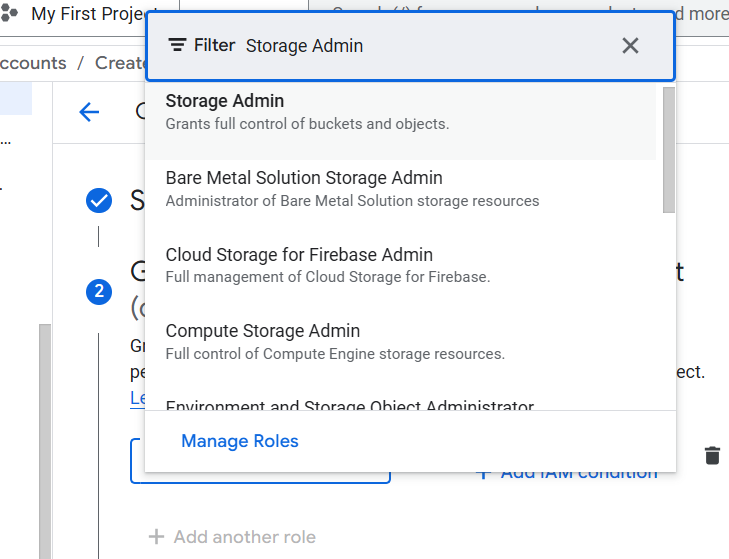


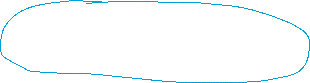


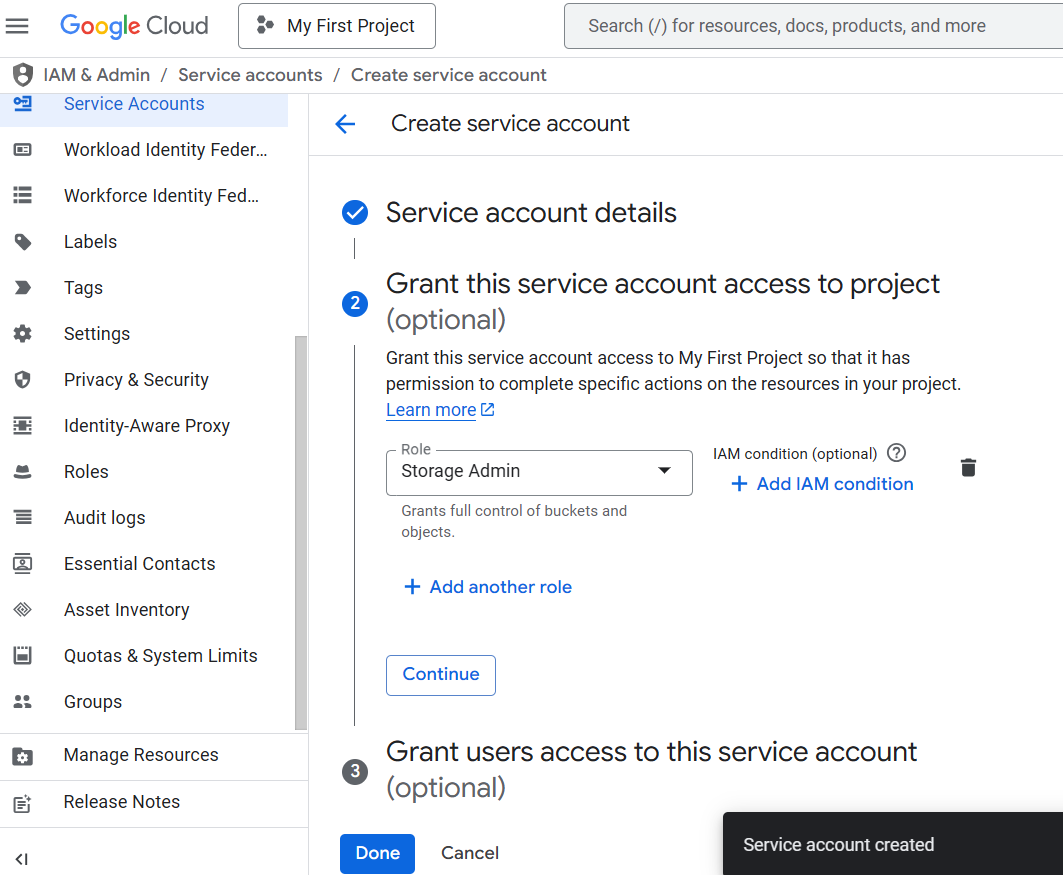
Give any name create and continue



Select storage role-storage admin

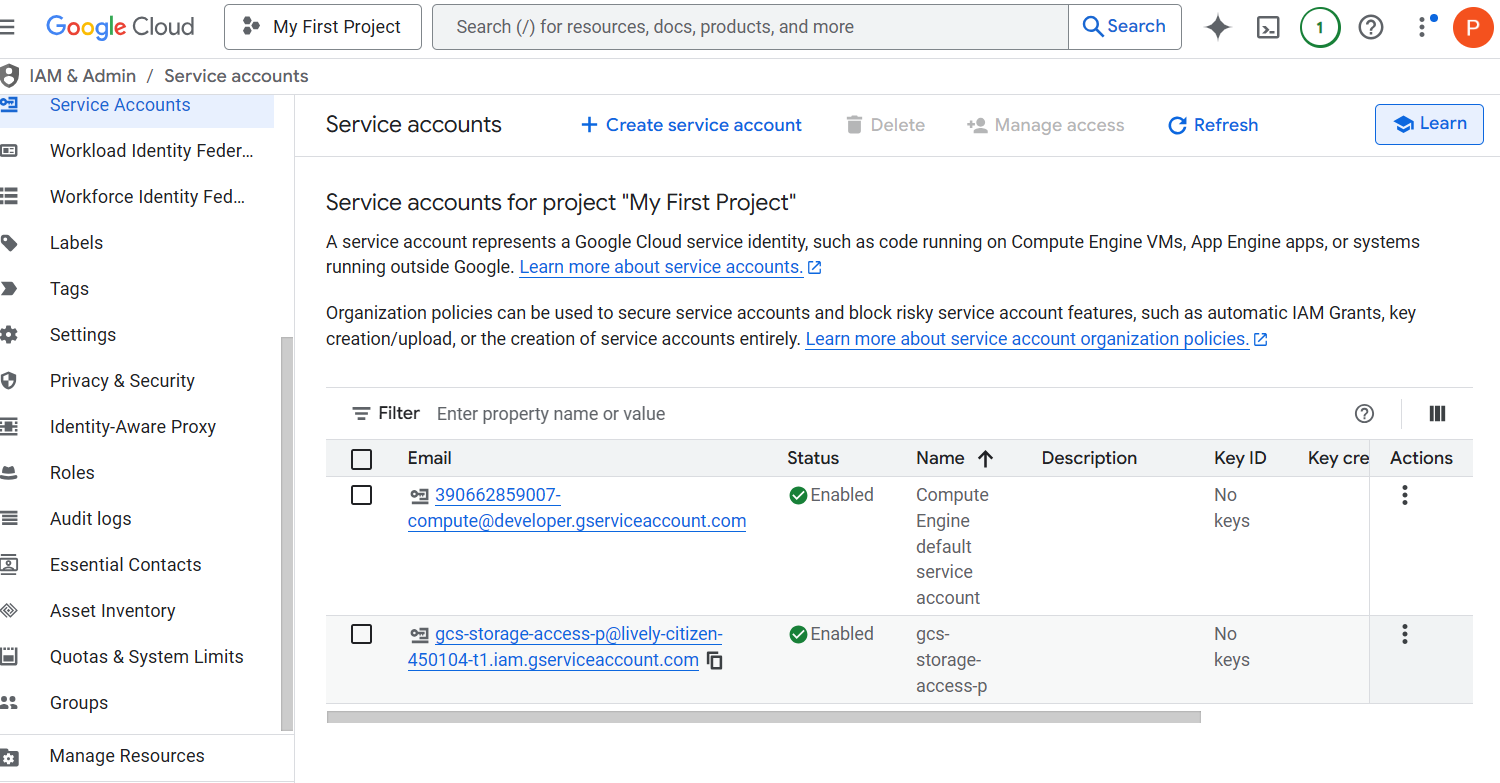




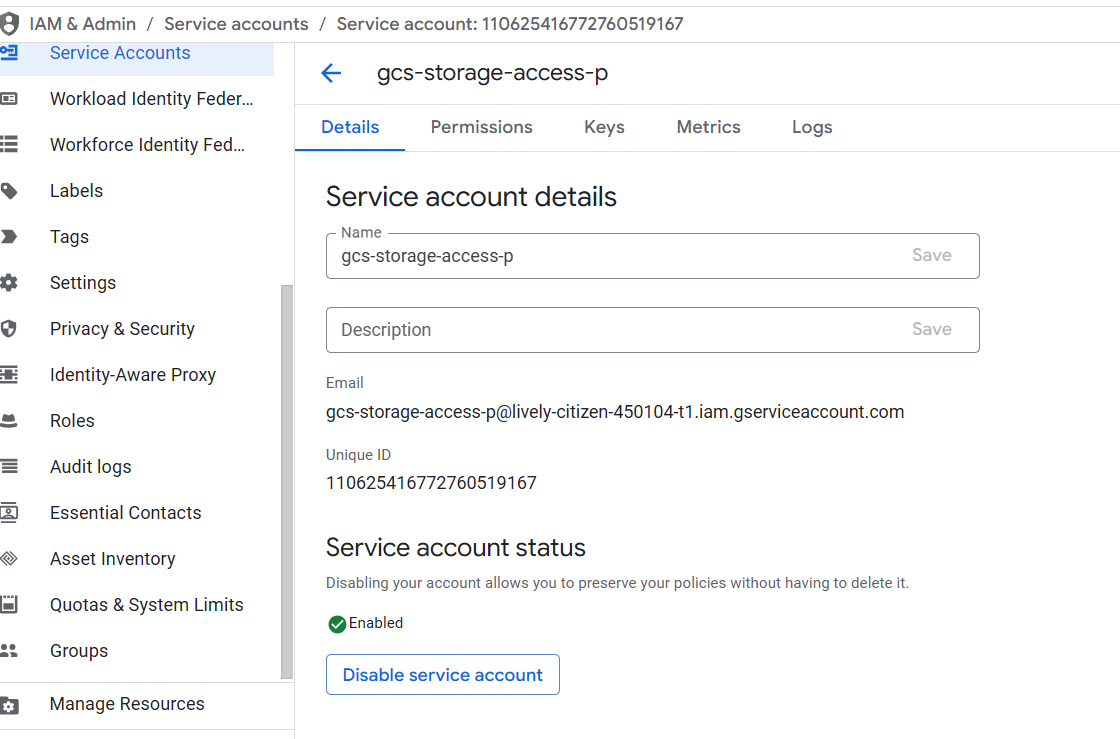




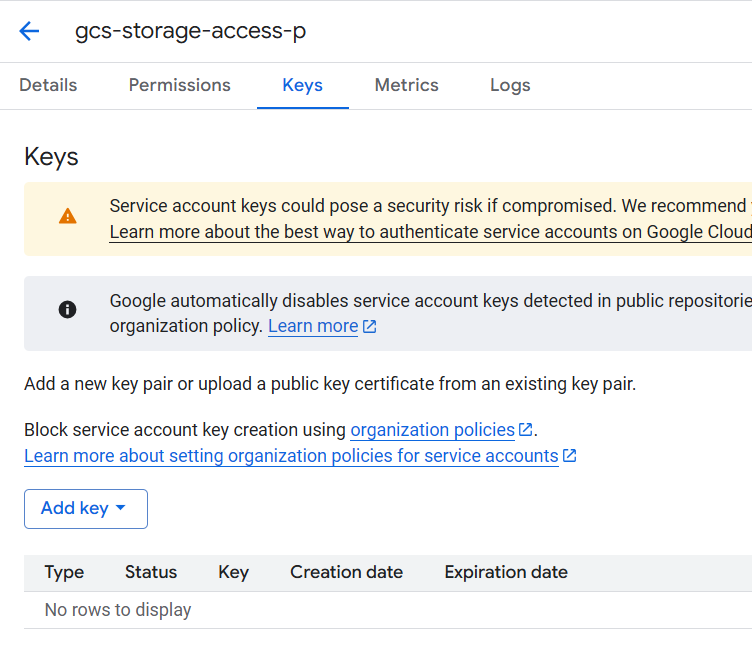
Now click on the storage service that you created

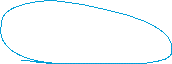


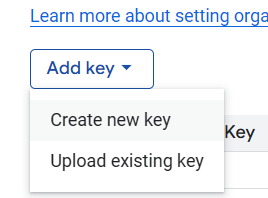




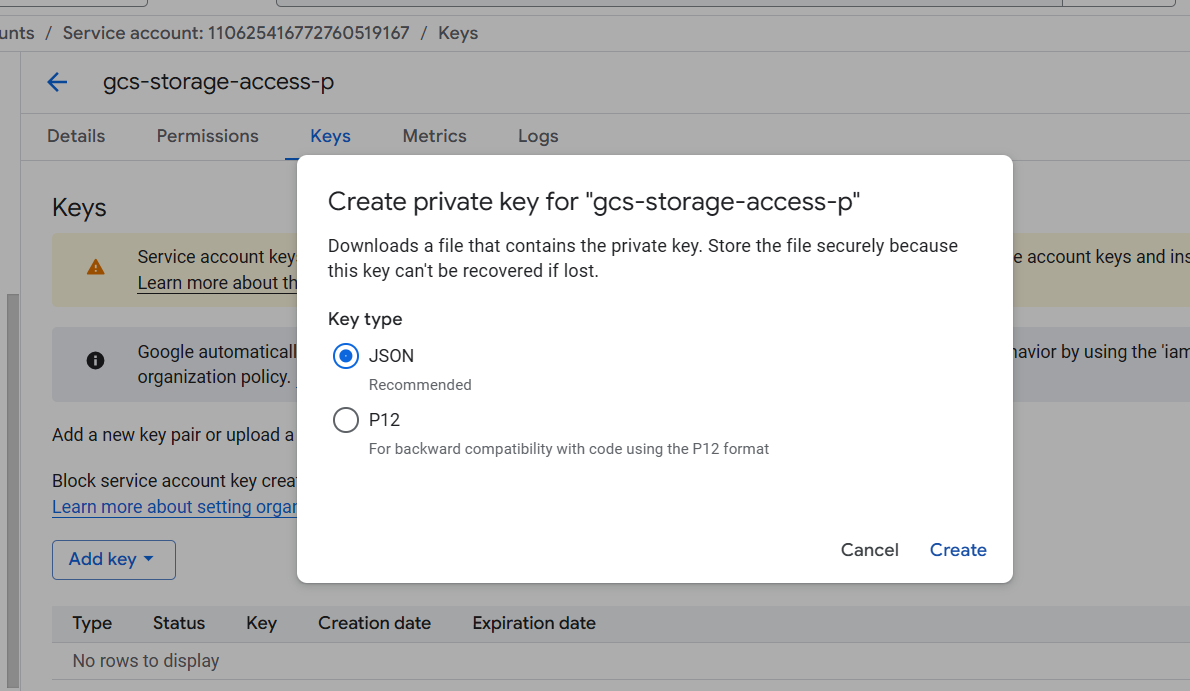






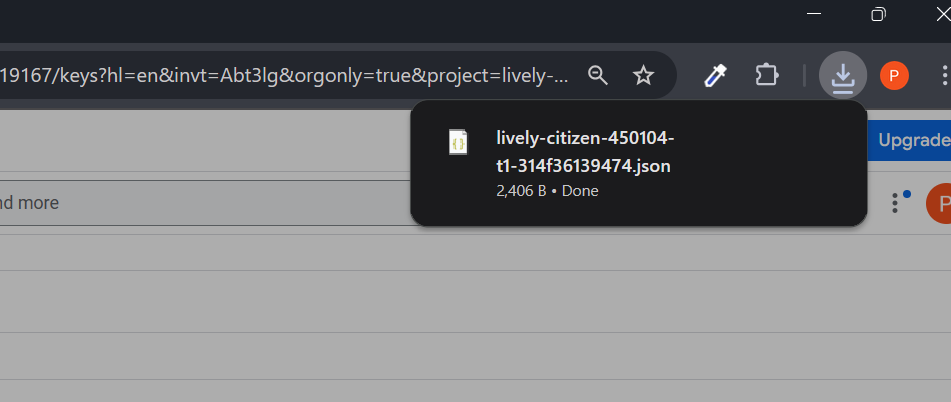




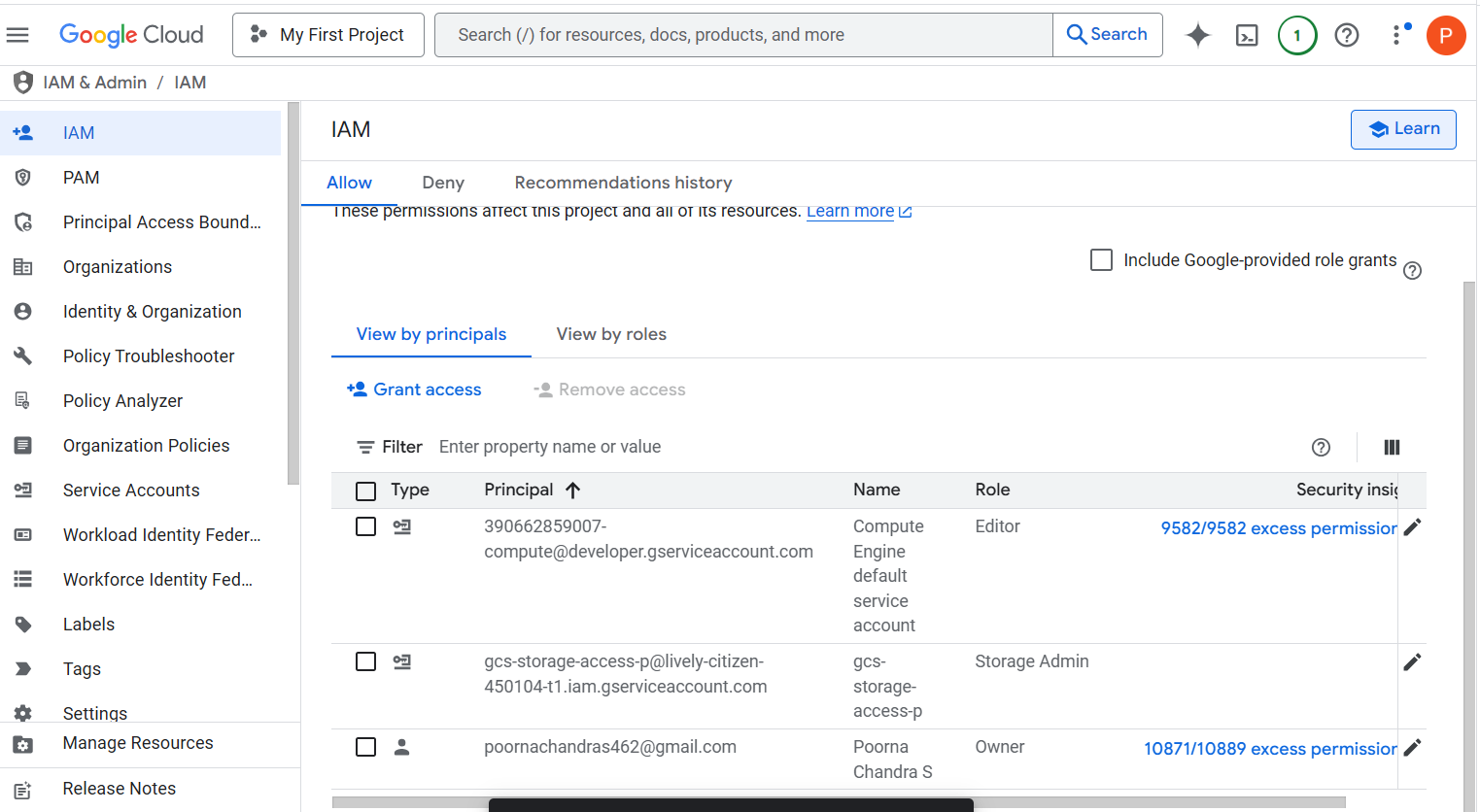




As soon as you click on create the key gets downloaded and add that json file in resource folder of you springboot project



Now we have not set permission to list bucket or store files



Ah, that’s a **permissions issue** — totally normal and easy to fix! 🔒

Your service account (gcs-storage-access-p@...) doesn't currently have the permission to **list buckets**, or maybe even upload files yet. Let’s grant the correct **roles/permissions** to that service account.

**✅ Step-by-Step: Grant Storage Access to Service Account**

**1. Go to IAM Page in GCP Console**

**2. Find the service account:**

gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com

**3. Click the pencil/edit icon ✏️**

**4. In the Roles section, click “Add Another Role” and choose:**

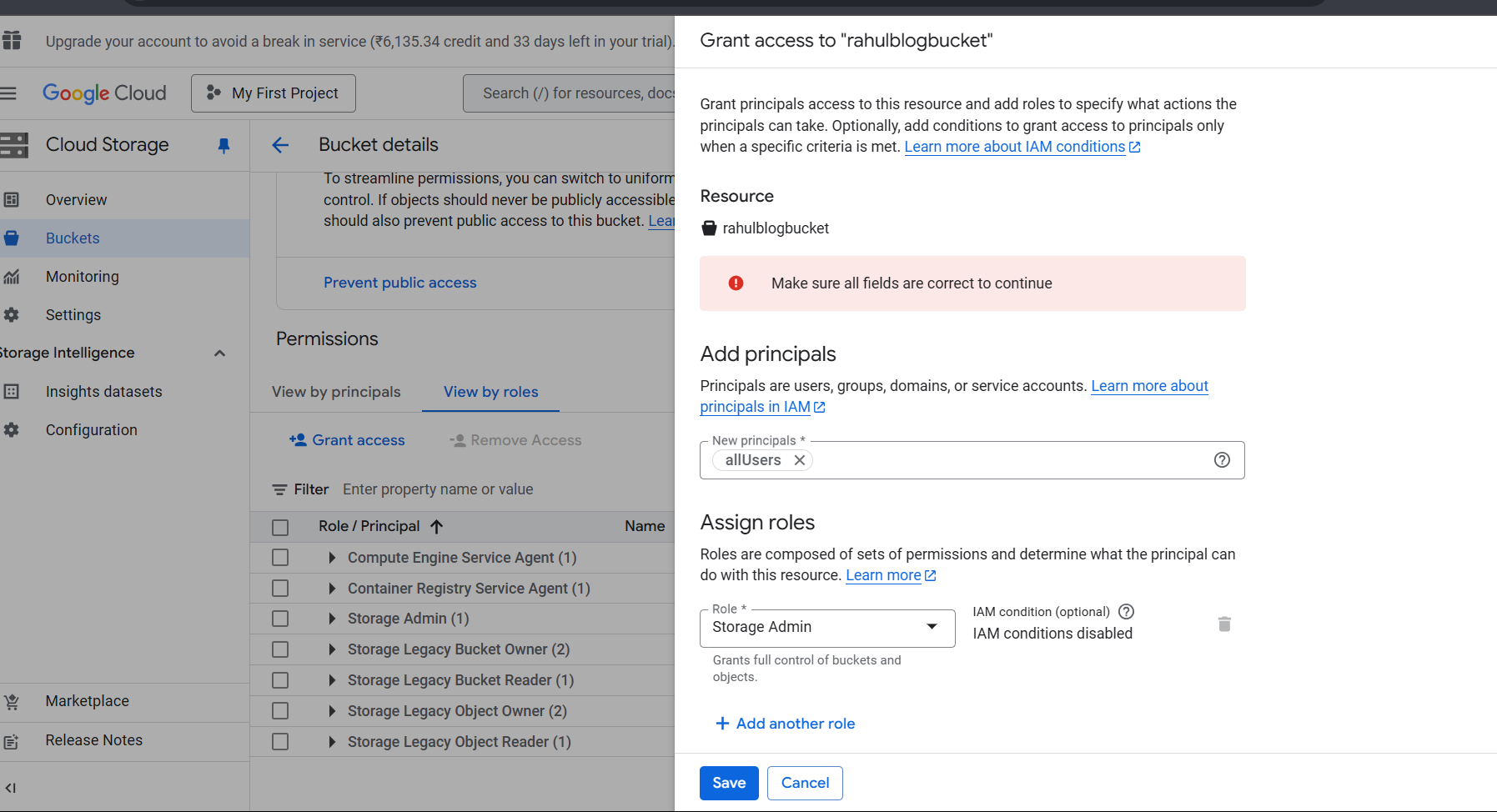
pgsql

CopyEdit

Cloud Storage > Storage Admin

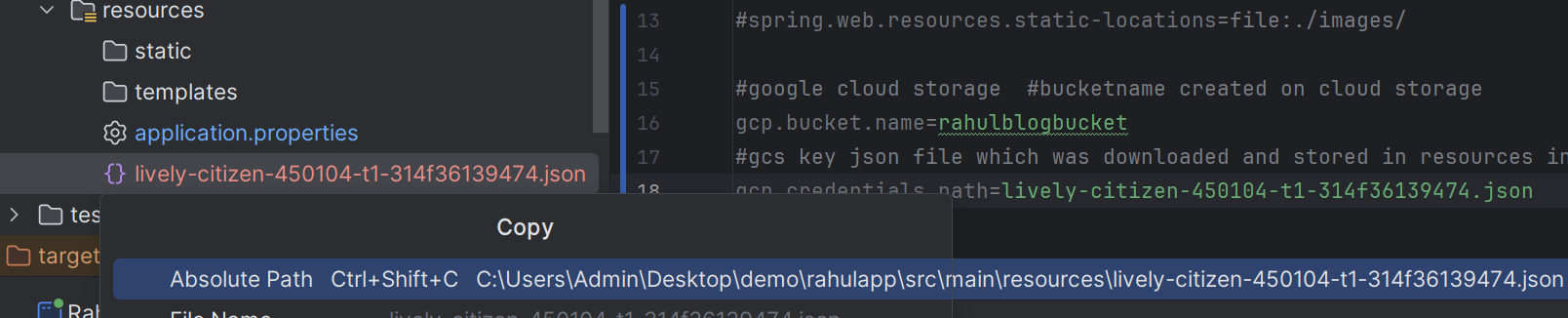
👉 This gives full read/write access (you can downgrade to just Storage Object Admin if you only want file upload/download and not bucket-level actions).

**5. Click Save.**





Now open google cloud shell



C:\Users\Admin\AppData\Local\Google\Cloud SDK>cd C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources\

**Authenticate**

C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources>gcloud auth activate-service-account --key-file=lively-citizen-450104-t1-314f36139474.json

Activated service account credentials for: [gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com]

Updates are available for some Google Cloud CLI components. To install them,

please run:

$ gcloud components update

To take a quick anonymous survey, run:

$ gcloud survey

**Get bucketList**

C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources>**gcloud storage buckets list**

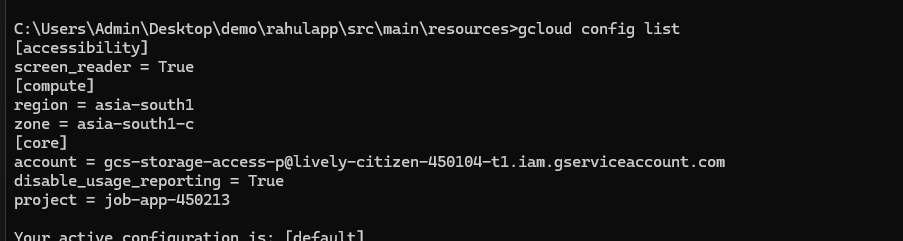
U might encounter this error…make sure you have created new project initially for this..

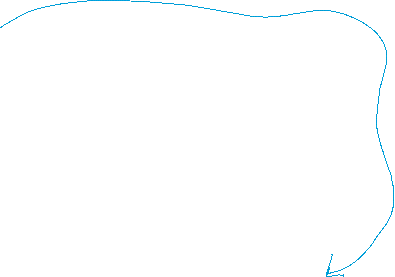
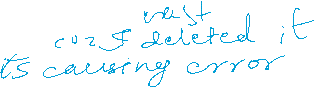
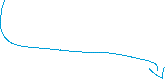
Or else the storage might be pointing to deleted project instance

*ERROR: (gcloud.storage.buckets.list) HTTPError 403: gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com does not have storage.buckets.list access to the Google Cloud project. Permission 'storage.buckets.list' denied on resource (or it may not exist). This command is authenticated as gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com which is the active account specified by the [core/account] property.*

**Get the list of configurations and check where the storage is created in which project**

C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources>**gcloud config list**

**



gcloud storage buckets list --project=another-project-id ..this command will get you the specific bucket list in that particular project



or else in config list the bucket name @Lvely will be mentioned just set it as default project

C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources>**gcloud config set project lively-citizen-450104-t1**

WARNING: [gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com] does not have permission to access projects instance [lively-citizen-450104-t1] (or it may not exist): Cloud Resource Manager API has not been used in p……..

Are you sure you wish to set property [core/project] to lively-citizen-450104-t1?

Do you want to continue (Y/n)? y

C:\Users\Admin\Desktop\demo\rahulapp\src\main\resources>**gcloud storage buckets list**

---

acl:

- entity: project-owners-390662859007

projectTeam:

projectNumber: '390662859007'

team: owners

role: OWNER

- entity: project-editors-390662859007

projectTeam:

projectNumber: '390662859007'

team: editors

role: OWNER

- entity: project-viewers-390662859007

projectTeam:

projectNumber: '390662859007'

team: viewers

role: READER

creation\_time: 2025-04-04T17:06:43+0000

default\_storage\_class: STANDARD

generation: 1743786403559384449

location: US

location\_type: multi-region

metageneration: 3

name: rahulblogbucket

public\_access\_prevention: inherited

rpo: DEFAULT

soft\_delete\_policy:

effectiveTime: '2025-04-04T17:06:43.832000+00:00'

retentionDurationSeconds: '604800'

storage\_url: gs://rahulblogbucket/

uniform\_bucket\_level\_access: false

update\_time: 2025-04-04T17:09:25+0000

==============================================================================================================================================================

Here's a complete Spring Boot application using **Spring MVC, JPA, Lombok, PostgreSQL**, and **Google Cloud Storage (GCS) for image uploads**.

**Steps**

1. **Create a Spring Boot Application** (Maven)
2. **Add Dependencies**
3. **Create Entity, DTO, Repository, Service, and Controller**
4. **Implement Google Cloud Storage File Upload**
5. **Test APIs**

**1. Add Dependencies in pom.xml**

xml

CopyEdit

<dependencies>

<!-- Spring Boot Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- PostgreSQL Driver -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

</dependency>

<!-- Lombok (Auto-generate Getters/Setters) -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok-mapstruct-binding</artifactId>

<version>0.2.0</version>

<scope>compile</scope>

</dependency>

<!-- Spring Boot Validation -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- Google Cloud Storage -->

<dependency>

<groupId>com.google.cloud</groupId>

<artifactId>google-cloud-storage</artifactId>

<version>2.26.0</version>

</dependency>

</dependencies>

**2. Configure application.properties**

properties

CopyEdit

server.port=8080

# PostgreSQL Configuration

spring.datasource.url=jdbc:postgresql://localhost:5432/ecommerce\_db

spring.datasource.username=your\_db\_user

spring.datasource.password=your\_db\_password

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Google Cloud Storage Config

gcp.bucket.name=your-bucket-name

gcp.credentials.path=classpath:gcp-key.json

**3. Create the Product Entity**

java

CopyEdit

package com.example.ecommerce.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String imageUrl;

private Double price;

}

**4. Create ProductDTO**

java

CopyEdit

package com.example.ecommerce.dto;

import lombok.Data;

import org.springframework.web.multipart.MultipartFile;

@Data

public class ProductDTO {

private String title;

private Double price;

private MultipartFile image; // To handle file uploads

}

**5. Create ProductRepository**

java

CopyEdit

package com.example.ecommerce.repository;

import com.example.ecommerce.model.Product;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface ProductRepository extends JpaRepository<Product, Long> {

}

**6. Implement Google Cloud Storage Service**

java

CopyEdit

package com.example.ecommerce.service;

import com.google.cloud.storage.\*;

import lombok.RequiredArgsConstructor;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.UUID;

@Service

@RequiredArgsConstructor

public class CloudStorageService {

@Value("${gcp.bucket.name}")

private String bucketName;

private final Storage storage = StorageOptions.getDefaultInstance().getService();

public String uploadFile(MultipartFile file) throws IOException {

String fileName = UUID.randomUUID() + "-" + file.getOriginalFilename();

BlobId blobId = BlobId.of(bucketName, fileName);

BlobInfo blobInfo = BlobInfo.newBuilder(blobId).setContentType(file.getContentType()).build();

storage.create(blobInfo, file.getBytes());

return String.format("https://storage.googleapis.com/%s/%s", bucketName, fileName);

}

}

**7. Create ProductService**

java

CopyEdit

package com.example.ecommerce.service;

import com.example.ecommerce.dto.ProductDTO;

import com.example.ecommerce.model.Product;

import com.example.ecommerce.repository.ProductRepository;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.List;

@Service

@RequiredArgsConstructor

public class ProductService {

private final ProductRepository productRepository;

private final CloudStorageService cloudStorageService;

public List<Product> getAllProducts() {

return productRepository.findAll();

}

public Product createProduct(ProductDTO productDTO) throws IOException {

String imageUrl = cloudStorageService.uploadFile(productDTO.getImage());

Product product = Product.builder()

.title(productDTO.getTitle())

.price(productDTO.getPrice())

.imageUrl(imageUrl)

.build();

return productRepository.save(product);

}

public void deleteProduct(Long id) {

productRepository.deleteById(id);

}

}

**8. Create ProductController**

java

CopyEdit

package com.example.ecommerce.controller;

import com.example.ecommerce.dto.ProductDTO;

import com.example.ecommerce.model.Product;

import com.example.ecommerce.service.ProductService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.List;

@RestController

@RequestMapping("/api/products")

@RequiredArgsConstructor

public class ProductController {

private final ProductService productService;

@GetMapping

public List<Product> getAllProducts() {

return productService.getAllProducts();

}

@PostMapping

public ResponseEntity<Product> createProduct(@ModelAttribute ProductDTO productDTO) {

try {

Product savedProduct = productService.createProduct(productDTO);

return ResponseEntity.ok(savedProduct);

} catch (IOException e) {

return ResponseEntity.internalServerError().build();

}

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {

productService.deleteProduct(id);

return ResponseEntity.noContent().build();

}

}

**9. Run and Test APIs**

1. Start PostgreSQL and ensure ecommerce\_db is created.
2. Run the Spring Boot application.
3. **Test API Endpoints using Postman or cURL**
   * **Upload Product with Image** (Use form-data in Postman)

yaml

CopyEdit

POST /api/products

Body:

- title: "Laptop"

- price: 1200

- image: (Select Image File)

* + **Get All Products**

bash

CopyEdit

GET /api/products

* + **Delete a Product**

bash

CopyEdit

DELETE /api/products/{id}

**10. Use Image URLs on Frontend**

Each product response includes an imageUrl, which you can directly use in <img> tags on the frontend:

html

CopyEdit

<img src="https://storage.googleapis.com/your-bucket-name/file-name.jpg" alt="Product Image" />

**🎯 Features Implemented**

✅ Spring Boot MVC  
✅ PostgreSQL Database  
✅ JPA + Repository Pattern  
✅ Lombok for Clean Code  
✅ Google Cloud Storage File Upload  
✅ DTOs for File Handling  
✅ Fully Working CRUD APIs

Would you like enhancements like **pagination, search, or JWT authentication**? 🚀

4o

**You said:**

first tell me how to setup bucket on gcp and get # Google Cloud Storage Config gcp.bucket.name=your-bucket-name gcp.credentials.path=classpath:gcp-key.json

**ChatGPT said:**

**Step-by-Step Guide to Setting Up a Google Cloud Storage (GCS) Bucket and Getting Credentials**

**Step 1: Create a Google Cloud Project**

1. Go to Google Cloud Console.
2. Click on the **Project Dropdown** (top left) and select **New Project**.
3. Give your project a **name** (e.g., ecommerce-storage).
4. Click **Create** and wait for it to be ready.

**Step 2: Enable Cloud Storage API**

1. In the **Google Cloud Console**, go to **APIs & Services** → **Library**.
2. Search for **Cloud Storage** and click on it.

**Choose: Cloud Storage API**

* **Name:** Cloud Storage API
* **Description:** "Google Cloud Storage is a RESTful service for storing and accessing your data on Google's infrastructure."
* **Why?** This is the core API required for using Google Cloud Storage with your backend.

1. Click **Enable**.

**Step 3: Create a Cloud Storage Bucket**

1. Go to **Storage** → **Buckets**.
2. Click **Create Bucket**.
3. Enter a unique **Bucket Name** (e.g., my-ecommerce-bucket).
4. Choose a **Storage Class** (Standard is fine).
5. Choose a **Location** (e.g., us-central1).
6. **Access Control:**
   * If you want public images: Select **Fine-grained** access.
   * If private: Select **Uniform** access.
7. Click **Create**.

**Step 4: Generate Service Account Credentials**

1. Go to **IAM & Admin** → **Service Accounts**.
2. Click **Create Service Account**.
3. Enter **Service Account Name** (e.g., gcs-storage-access).
4. Click **Create and Continue**.
5. Assign **Role** → Select **Storage Admin**.
6. Click **Done**.

**Step 5: Download JSON Credentials**

1. In **IAM & Admin → Service Accounts**, find your service account.
2. Click on it → Go to **Keys** tab.
3. Click **Add Key** → Select **JSON**.
4. Click **Create**, and it will download a JSON file (e.g., gcp-key.json).

What to Do Next: Generate the JSON Key

Go to the Service Accounts page in your GCP Console.

Click on the gcs-storage-access-p service account.

Go to the "Keys" tab.

Click “Add Key” → “Create New Key”.

Choose Key type: JSON, then click Create.

✅ It will download a JSON file like: gcs-storage-access-p-xxxxxxxx.json.

📁 What to Do With the JSON File

Rename it (optional): e.g., gcp-key.json.

Move it to your Spring Boot project under:

src/main/resources/gcp-key.json

Then in application.properties, add:

properties

Copy

Edit

gcp.bucket.name=your-bucket-name

gcp.credentials.path=classpath:gcp-key.json

**Step 6: Configure Spring Boot**

Move the downloaded gcp-key.json file to src/main/resources/.

**Set Up application.properties**

properties

CopyEdit

# Google Cloud Storage Config

gcp.bucket.name=my-ecommerce-bucket

gcp.credentials.path=classpath:gcp-key.json

**Step 7: Verify and Test**

Run this command to check if your setup works:

sh

CopyEdit

gcloud auth activate-service-account --key-file=src/main/resources/gcp-key.json

Then, list buckets:

sh

CopyEdit

gcloud storage buckets list

If you see your bucket name, you're good to go! 🚀

Would you like help with making the bucket **publicly accessible** or generating **signed URLs** for private access?

/////////last but not the list if any permission errors

Ah, that’s a **permissions issue** — totally normal and easy to fix! 🔒

Your service account (gcs-storage-access-p@...) doesn't currently have the permission to **list buckets**, or maybe even upload files yet. Let’s grant the correct **roles/permissions** to that service account.

**✅ Step-by-Step: Grant Storage Access to Service Account**

**1. Go to IAM Page in GCP Console**

**2. Find the service account:**

gcs-storage-access-p@lively-citizen-450104-t1.iam.gserviceaccount.com

**3. Click the pencil/edit icon ✏️**

**4. In the Roles section, click “Add Another Role” and choose:**

pgsql

CopyEdit

Cloud Storage > Storage Admin

👉 This gives full read/write access (you can downgrade to just Storage Object Admin if you only want file upload/download and not bucket-level actions).

**5. Click Save.**