1. Quick overview (what you'll build)

A Spring Boot REST service that can:

- upload files to a GCS bucket,
- list objects in a bucket,
- download objects,
- delete objects,
- generate short-lived signed URLs for direct client upload/download.

We'll use the official Google Cloud Java client library (or Spring Cloud GCP helper) from a Gradle Spring Boot project. GitHub+1

---

2. Prerequisites

- Google Cloud Project with billing enabled.
- gcloud CLI and gsutil installed & authenticated.
- JDK 17+ (or JDK 11), Gradle, and IDE.
- A service account with appropriate Storage roles (at least **Storage Object Admin** for full object-level ops, or use least-privilege roles you need).
- Basic Spring Boot knowledge.

Official Cloud Storage docs (quickstarts & samples) are useful references. Google Cloud Documentation

---

3. GCP setup (commands)

1. Create a GCP project (or use existing):

```
gcloud projects create my-gcs-demo --name="My GCS Demo Project"

gcloud config set project my-gcs-demo
```

2. Enable the Cloud Storage API:

```
gcloud services enable storage.googleapis.com
```

3. Create a bucket (global unique name). Example — multi-region:

```
gcloud storage buckets create gs://my-gcs-demo-bucket --uniform-bucket-level-access
```

4. Create a service account and give it minimal permissions (Storage Object Admin if you want full object ops):

```
gcloud iam service-accounts create spring-gcs-sa --display-name="spring gcs sa"
```

```
gcloud projects add-iam-policy-binding my-gcs-demo \
 --member="serviceAccount:spring-gcs-sa@my-gcs-demo.iam.gserviceaccount.com" \
 --role="roles/storage.objectAdmin"
```

5. Create & download JSON key (for local dev only — prefer Workload Identity / ADC in prod):

```
gcloud iam service-accounts keys create ./spring-gcs-sa-key.json \
 --iam-account=spring-gcs-sa@my-gcs-demo.iam.gserviceaccount.com
```

---

4. Project setup (Gradle + Spring Boot)

Create a new Gradle Spring Boot project (or add to existing).

build.gradle (snippet for Gradle Kotlin DSL or plain Groovy — show essential deps):

```
plugins {
  id 'org.springframework.boot' version '3.2.0' // pick appropriate version
  id 'io.spring.dependency-management' version '1.1.0'
  id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
  mavenCentral()
}

dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-web'

  // Option A: Official Google Cloud Storage Java client
  implementation 'com.google.cloud:google-cloud-storage:2.28.0' // check for latest
```

```
    // Option B: Spring Cloud GCP helper (optional - provides ResourceLoader "gs:" support)
    implementation 'org.springframework.cloud:spring-cloud-gcp-starter-storage:4.0.0'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Use either the Google client library directly or Spring Cloud GCP starter. Spring Cloud GCP offers Spring abstractions (ResourceLoader with gs: URIs) and integrates nicely. GitHub+1

---

5. Configuration (application.properties)

```
# GCS settings
app.gcs.bucket=demo-gcs-nov-4-2025
app.gcs.project-id=keen-ally-473016-s9
# Set to absolute or relative path to JSON key for local dev. Leave empty to rely on ADC/GOOGLE_APPLICATION_CREDENTIALS.
app.gcs.credentials-file=./keen-ally-473016-s9-af8d5a8236ff.json

# Server port
server.port=8080
```

If using google-cloud-storage directly you can also configure project id in code or rely on ADC (Application Default Credentials).

---

6. Core service: StorageService (using google-cloud-storage)

Below is a simple GcsStorageService showing upload, download, list, delete, and signed URL generation using com.google.cloud.storage.Storage.

```
package com.example.gcsdemo.service;

import com.google.cloud.storage.*;
import com.google.cloud.storage.Storage.BlobListOption;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
```

```
import java.net.URL;
import java.nio.channels.Channels;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class GcsStorageService {

  private final Storage storage;
  private final String bucketName;

  public GcsStorageService(Storage storage,
                @Value("${app.gcs.bucket}") String bucketName) {
    this.storage = storage;
    this.bucketName = bucketName;
  }

  // Upload
  public String upload(MultipartFile file, String objectName) throws IOException {
    BlobId blobId = BlobId.of(bucketName, objectName);
    BlobInfo blobInfo = BlobInfo.newBuilder(blobId)
        .setContentType(file.getContentType())
        .build();
    storage.create(blobInfo, file.getInputStream());
    return objectName;
  }

  // Download (returns byte[])
  public byte[] download(String objectName) {
    Blob blob = storage.get(BlobId.of(bucketName, objectName));
```

```
        if (blob == null) return null;
        return blob.getContent();
    }

    // List objects
    public List<String> listObjects() {
        List<String> names = new ArrayList<>();
        Page<Blob> blobs = storage.list(bucketName, BlobListOption.currentDirectory());
        for (Blob b : blobs.iterateAll()) {
            names.add(b.getName());
        }
        return names;
    }

    // Delete object
    public boolean delete(String objectName) {
        return storage.delete(BlobId.of(bucketName, objectName));
    }

    // Generate signed URL (v4) for download (valid for `duration` minutes)
    public URL generateSignedUrl(String objectName, int durationMinutes) {
        BlobInfo blobInfo = BlobInfo.newBuilder(BlobId.of(bucketName, objectName)).build();
        URL signedUrl = storage.signUrl(
            blobInfo,
            durationMinutes,
            TimeUnit.MINUTES,
            Storage.SignUrlOption.withV4Signature());
        return signedUrl;
    }
}
```

How Storage storage is created: you can autowire the client via Spring configuration:

```
import com.google.cloud.storage.Storage;
import com.google.cloud.storage.StorageOptions;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class GcsConfig {
  @Bean
  public Storage storage() {
    return StorageOptions.getDefaultInstance().getService();
  }
}
```

This uses ADC (Application Default Credentials) so make sure GOOGLE_APPLICATION_CREDENTIALS is set for local dev, or that the environment has proper credentials (Cloud Run, GKE service account, etc.).

Official code samples for upload/download are in the Cloud Storage docs and the Java client repo. Google Cloud Documentation+1

---

7. Controller endpoints (example)

```
package com.example.gcsdemo.controller;

import com.example.gcsdemo.service.GcsStorageService;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequestMapping("/api/files")
public class FileController {
```

```
    private final GcsStorageService gcsService;

    public FileController(GcsStorageService gcsService) {
        this.gcsService = gcsService;
    }

    @PostMapping("/upload")
    public ResponseEntity<String> upload(@RequestParam("file") MultipartFile file) throws
Exception {
        String objectName = System.currentTimeMillis() + "_" + file.getOriginalFilename();
        gcsService.upload(file, objectName);
        return ResponseEntity.ok(objectName);
    }

    @GetMapping("/download/{name}")
    public ResponseEntity<byte[]> download(@PathVariable("name") String name) {
        byte[] data = gcsService.download(name);
        if (data == null) return ResponseEntity.notFound().build();
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + name +
"\"")
                .contentType(MediaType.APPLICATION_OCTET_STREAM)
                .body(data);
    }

    @GetMapping("/list")
    public ResponseEntity<?> list() {
        return ResponseEntity.ok(gcsService.listObjects());
    }

    @DeleteMapping("/{name}")
    public ResponseEntity<?> delete(@PathVariable String name) {
```

```
    boolean removed = gcsService.delete(name);
    return removed ? ResponseEntity.noContent().build() : ResponseEntity.notFound().build();
  }

  @GetMapping("/signed-url/{name}")
  public ResponseEntity<String> signedUrl(@PathVariable String name) {
    return ResponseEntity.ok(gcsService.generateSignedUrl(name, 15).toString());
  }
}
```

---

8. Test locally (curl)

Upload:

curl -F "file=@/path/to/local.jpg" http://localhost:8080/api/files/upload

List:

curl http://localhost:8080/api/files/list

Download:

curl -o out.jpg http://localhost:8080/api/files/download/<objectName>

Get signed URL:

curl http://localhost:8080/api/files/signed-url/<objectName>

---

9. Best practices & production considerations

- **Use ADC & Workload Identity**: Don't ship JSON keys to production. Use Workload Identity on GKE, Cloud Run service accounts, or IAM roles for VMs. Google Cloud Documentation
- **Least-privilege IAM**: Assign only required roles (storage.objectViewer, storage.objectCreator, etc.) not owner.
- **Uniform bucket-level access**: use it to simplify IAM. (we used this in creation step).
- **Object versioning**: enable if you need to recover from accidental deletes/overwrites.
- **Bucket lifecycle rules**: set lifecycle rules to auto-delete/archive old objects (e.g., move to Nearline/Coldline or delete after X days).

- **Signed URLs & Signed Policy Docs**: For direct client uploads/downloads, generate signed URLs from your backend rather than exposing credentials. Use V4 signed URLs for wider compatibility. Google Cloud Documentation
- **CORS**: If you do direct browser uploads, configure CORS for your bucket.
- **Large files**: Use resumable uploads for large objects. The Java client and gsutil support resumable uploads. GitHub
- **Encryption & KMS**: Use CMEK (Customer-managed encryption keys) when required.
- **Logging & Monitoring**: Enable access logs and integrate with Cloud Logging/Monitoring for auditing.
- **Performance**: Use appropriate storage class (Standard / Nearline / Coldline) and choose bucket location (region/multi-region) according to latency and cost needs. Google Cloud Documentation

---

10. Industry use cases (examples)

- **User uploads** (profile pics, documents) — store objects, generate signed URLs for direct upload.
- **Static assets / CDN-backed hosting** — host public static assets and front them with Cloud CDN.
- **Backups & exports** — application backups or periodic DB exports to Coldline for cost savings.
- **ML artifacts** — model binaries, training data stored for Dataflow/AI pipelines.
- **Data lakes** — raw/processed data in buckets with lifecycle transitions.

---

11. Troubleshooting tips

- 403 or Access Denied: check IAM role and uniform bucket-level access. Ensure ADC/service account is the one you expect.
- NoSuchBucket: verify bucket name and project (bucket names are global).
- JSON key not found: ensure GOOGLE_APPLICATION_CREDENTIALS points to the JSON file.
- Large files/timeouts: use resumable uploads or chunking and increase timeouts in HTTP client.
