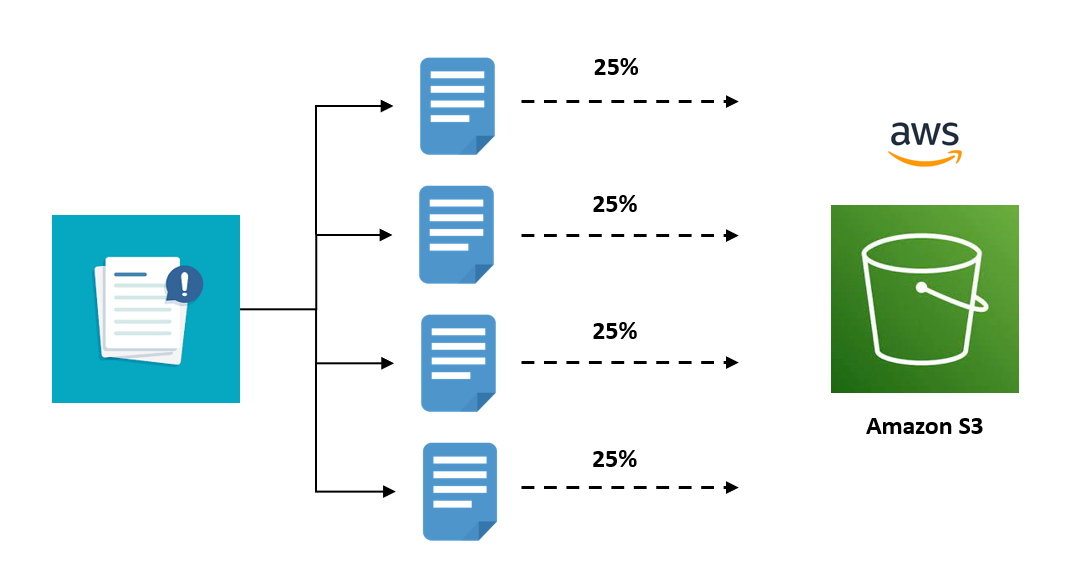
## **S3 Multipart Upload Project**

This project demonstrates how to upload a file to an Amazon S3 private bucket using the AWS CLI and multipart upload.

### **1. Introduction**

Hi, I’m Paramesh, and this repository showcases a simple yet effective way to **upload files to a private Amazon S3 bucket** using the AWS CLI’s multipart upload support. In this write-up, I’ll walk through every step I took, from initial setup to testing.



### **2. Project Setup**

**Cloning the repository**

git clone https://github.com/Paramesh05-svg/S3-project.git

cd S3-project

Here, you'll find:

* README.md: overview instructions
* Config.file: to store AWS credentials and bucket info
* file\_path.json: sample file path metadata

### **3. Prerequisites**

Before I began, I ensured I had the following ready:

1. **AWS account** with an S3 bucket (set to private).
2. **AWS CLI** installed and configured:  
    aws configure
3. S3 Bucket: Create a private S3 bucket.
4. An **IAM user** with permissions like: s3:PutObject, s3:AbortMultipartUpload, and s3:ListMultipartUploadParts.

### **4. Configuration**

I copied Config.file, updated it, and renamed it for clarity—like this aws\_config.json:

{

"bucket": "my-private-bucket-name",

"region": "us-east-1",

"file\_key": "uploads/my\_big\_file.zip",

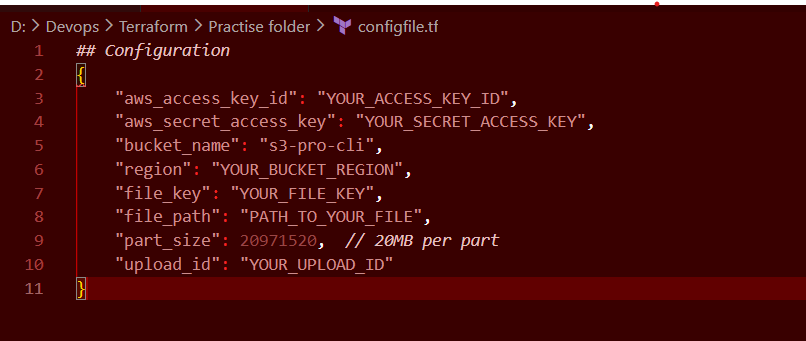
"file\_path": "/path/to/my\_big\_file.zip"

}

This configuration cleanly separates the S3 bucket name, region, object key, and the local file path, making the code reusable and easy to manage.

**Note:** I intentionally omitted **AWS Access Key ID** and **Secret Access Key** from this file for security reasons. These credentials should be securely stored using environment variables, AWS credentials files, or secret management services, and **must never be hardcoded or shared** in source files or version control systems.

I’ve also included a sample image showcasing reusable code that utilizes this configuration format, further promoting secure and maintainable development practices.



### **5. Multipart Upload with AWS CLI**

I rely on AWS CLI’s built-in support for multipart uploads:

aws s3 cp "/path/to/my\_big\_file.zip" s3://my-private-bucket-name/uploads/ \

--region us-east-1 --storage-class STANDARD

When files exceed ~100 MB, AWS CLI automatically handles multipart uploads—splitting the file into parts, uploading each in parallel, and stitching them back together. I didn't have to write additional code for that, keeping things clean and efficient.

### **6. How It Works Under the Hood**

AWS CLI manages multipart uploads through these steps:

1. **Create** a multipart upload session.
2. **Split** the file into multiple 20+ MB parts.
3. **Upload** each part independently.
4. **Complete** the upload by assembling parts into one object.
5. If interrupted, remaining parts can be resumed or aborted.

This enables reliable uploads, especially for large files.

### **7. Uploading a File: Step-by-Step**

Here’s a quick demo of my upload process:

1. **1.Configure AWS CLI**:  
     
   aws configure
2. **Run the upload**:  
     
   aws s3 cp "$FILE\_PATH" s3://$BUCKET\_NAME/$FILE\_KEY
3. **Verify** the upload:  
   * Via AWS Console (confirm file appears in the bucket).
   * Or using aws s3 ls s3://.../uploads/.

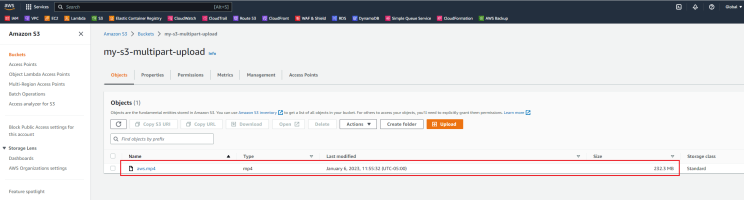
I have attached the image which shows the overall upload of all parts of the file.



### **8. Why Multipart Upload Matters**

I appreciate multipart uploads because they:

* **Enable resume** if the upload breaks mid-way.
* **Improve throughput** by uploading parts in parallel.
* **Bypass single-request limits**, allowing uploads of files over 5 GB effortlessly.



Finally , we can find the complete file on the designated private bucket.

### **9. Security and Privacy**

To keep things secure:

* I use a **private bucket**; no public reads/writes.
* My IAM user has **least-privilege**—only multipart upload permissions.
* Credentials are stored in ~/.aws/credentials and not pushed to GitHub, keeping them private.

### **10. Potential Enhancements**

In future versions, I’d like to add:

* **Resume support** via aws s3 mv or --no-overwrite.
* **Multipart status** outputs using aws s3api list-multipart-uploads.
* **Node.js or Python scripts** using AWS SDK to automate the process programmatically.

**11. When to Use Multipart Upload**

* Uploading files **larger than 100MB** (Amazon recommends multipart upload for files >100MB, and requires it for files >5GB).
* When you have a **slow or unstable network** connection and want the ability to retry failed parts.
* For uploading **video, large backups, zip archives**, and other heavy payloads.
* When you're working in **distributed systems** and want to split the upload process across different servers or threads.

### **12. Benefits of S3 Multipart Upload (Short Version)**

### Faster Uploads – Uploads large files in parallel parts for improved speed.

### Resumable – Can resume failed uploads without starting over.

### Memory Efficient – Uploads file parts without loading the full file into memory.

### Pause & Resume – Supports pausing and resuming uploads.

### Error Isolation – Only failed parts need to be re-uploaded.

### Data Integrity – Each part is validated with checksums.

### Recommended for Large Files – Ideal for files over 100MB; required for over 5GB.

### **13. Conclusion**

In creating this repo, my goals were simplicity and clarity. I achieved reliable multipart uploads with the AWS CLI and kept everything well-organized and secure. It’s a solid foundation to build more advanced functionality.