How does S3 Multi-Part Upload work? When should you use it?

Amazon S3 Multi-Part Upload allows you to upload large files in parts, which are then assembled by S3 into a single object. It improves upload performance, resiliency, and control over large file uploads.

How It Works

1. **Initiate Upload**
   You send a `CreateMultipartUpload` request to get an **UploadId**.
2. **Upload Parts (in parallel or sequence)**
   - You divide the file into **parts** (each part is typically 5MB to 5GB).
   - Each part is uploaded with `UploadPart` using the **UploadId**.
   - You can upload parts **in parallel** (good for speed).
3. **Complete Upload**
   - After all parts are uploaded, you call `CompleteMultipartUpload`.
   - S3 assembles them in order and creates the final object.
4. **Abort Upload (if needed)**
   - If upload fails or is canceled, you can call `AbortMultipartUpload` to avoid paying for partial uploads.

When Should we Use It?

Use Case	Why Use Multipart Upload
Uploading large files (>100 MB, especially >5 GB)	Standard upload fails or times out. Multi-part is more resilient.
Unreliable network or long uploads	Each part is independent. Failures don't require restarting

Use Case	Why Use Multipart Upload
	the whole upload.
Improving performance	You can upload multiple parts in parallel (multi-threading, multiprocessing).
Automated pipelines or backup tools	Easily control, resume, and retry uploads in CI/CD or backup jobs.
Large log archives, backups, or DB dumps	Often very large files that benefit from chunked upload.

Advantages

- **Parallelism** = Faster uploads using threads/processes.
- **Resumability** = Recover from part failure without restarting.
- **Efficiency** = Only upload what's needed if interrupted.
- **Automation** = Works great with scripts using AWS SDK, Boto3, AWS CLI.

🛠️ Example (AWS CLI)

```
bash

aws s3api create-multipart-upload --bucket my-bucket --key bigfile.zip
aws s3api upload-part --bucket my-bucket --key bigfile.zip --part-number 1 --
upload-id <ID> --body part1
aws s3api complete-multipart-upload --bucket my-bucket --key bigfile.zip --
upload-id <ID> --multipart-upload file://parts.json
```

📌 Key Points to Remember

Use S3 Multi-Part Upload when:

- Uploading **files >100MB**
- Need **faster, parallel uploads**
- Require **upload resiliency and retries**
- Working with **automated systems** that generate large artifacts/logs/backups
- **Mandatory** for objects larger than **5 GB**

Node.js (AWS SDK v3)

javascript

```javascript
// npm install @aws-sdk/client-s3 @aws-sdk/lib-storage
const { S3Client } = require("@aws-sdk/client-s3");
const { Upload } = require("@aws-sdk/lib-storage");
const fs = require("fs");

const s3 = new S3Client({ region: "us-east-1" });

const uploadFile = async () => {
  const fileStream = fs.createReadStream("./large-video.mp4");

  const upload = new Upload({
    client: s3,
    params: {
      Bucket: "your-bucket-name",
      Key: "videos/large-video.mp4",
      Body: fileStream,
    },
  });

  upload.on("httpUploadProgress", (progress) => {
    console.log(`Uploaded ${progress.loaded} bytes out of ${progress.total}`);
  });

  try {
    const result = await upload.done();
    console.log("Upload complete:", result);
  } catch (err) {
    console.error("Upload failed:", err);
  }
};

uploadFile();
```

> ✅ This approach uses the `@aws-sdk/lib-storage` which automatically handles multi-part uploads behind the scenes for large files.

Python (boto3)

python

```python
import boto3
from boto3.s3.transfer import TransferConfig

s3 = boto3.client('s3')

# Set multipart threshold to 50 MB
config = TransferConfig(multipart_threshold=50 * 1024 * 1024)

def upload_file():
    try:
        s3.upload_file(
            Filename='large-video.mp4',
            Bucket='your-bucket-name',
            Key='videos/large-video.mp4',
            Config=config,
            Callback=ProgressPercentage('large-video.mp4')
        )
        print("Upload complete.")
    except Exception as e:
        print("Upload failed:", e)

# Optional: Progress bar
import os, sys

class ProgressPercentage:
    def __init__(self, filename):
        self._filename = filename
        self._size = float(os.path.getsize(filename))
        self._seen_so_far = 0

    def __call__(self, bytes_amount):
        self._seen_so_far += bytes_amount
        percentage = (self._seen_so_far / self._size) * 100
        sys.stdout.write(f"\r{self._filename} {self._seen_so_far:.0f} /
{self._size:.0f} bytes ({percentage:.2f}%)")
        sys.stdout.flush()

upload_file()
```

✅ The `TransferConfig` and `upload_file` method handle the splitting and reassembling for you automatically.

Summary

Feature	Node.js	Python (boto3)
Library Used	`@aws-sdk/lib-storage`	`boto3.s3.transfer`
Handles retry/parts	✅	✅
Custom control	Yes (via low-level API)	Yes (via MultipartUpload API too)
Good for large files	✅	✅

What is the minimum size (in MB) for each part in an S3 Multi-Part Upload (except the last part)?

A) 1 MB
B) 5 MB
C) 10 MB
D) 50 MB

✅ **Correct Answer:** B — Minimum part size is **5 MB**, except for the last part.

Which of the following is a key benefit of using S3 Multi-Part Upload?

A) Reduces storage cost
B) Encrypts each part separately
C) Allows parallel upload of parts for faster and more reliable uploads
D) Automatically deletes incomplete uploads

✅ **Correct Answer:** C

When should you definitely use Multi-Part Upload for S3?

A) When uploading text files
B) When uploading files larger than 100 MB
C) When uploading files larger than 5 GB
D) When downloading files from S3

✅ **Correct Answer:** C — Multi-part upload is **required** for files larger than 5 GB.
