Web Service PeerLinkS exceeded its memory limit

An instance of your Web Service PeerLinkS exceeded its memory limit, which triggered an automatic restart. While restarting, the instance was temporarily unavailable.

This might have been caused by:

A memory leak in your application

A spike in incoming traffic

An undersized instance type for your use case

Recommended actions

Check your service's logs and metrics to help identify why the instance exceeded its memory limit.

If you find a memory leak, patch the leak and redeploy your service.

If you experienced a traffic spike, consider scaling your service to help handle future spikes.

Otherwise, your service might require more memory for your use case. To increase memory, upgrade your instance type.

If you take no action, your service might exceed its memory limit again, resulting in additional interruptions.

If you can't resolve this issue, please reach out to support@render.com.

Best,

The Render team

You can change notification settings for this service, along with default notification settings for your account/team.

© 2025 Render, 525 Brannan St #3

what this gmail means in simple word  
  
  
  
  
  
Got it! Here’s what that email means in **simple terms**:

**What happened**

* Your web service **PeerLinkS** used **more memory than allowed** on Render.
* Because of that, Render **restarted your service automatically** to protect the server.
* While it was restarting, your app was **temporarily unavailable**.

**Why this can happen**

1. **Memory leak** – your code is using more and more memory over time.
2. **Traffic spike** – suddenly a lot of users visited your site, using more memory.
3. **Small instance** – the server instance you chose might not have enough memory for your app.

**What you should do**

* Check your **logs** to see if your app crashed or used too much memory.
* If it’s a **memory leak**, fix the code.
* If it’s a **traffic spike**, consider **scaling your service** (bigger instance or more instances).
* If your app normally needs more memory, **upgrade your instance type**.

💡 **In short:** Your app tried to use too much memory, so Render restarted it. You need to **check logs and maybe upgrade memory or fix a leak**.

If you want, I can explain **why your P2P app might be using too much memory** and how to fix it for videos/uploads. Do you want me to do that?

so wht is the solution bcz i want to send video also  
  
Got it — since you want to **support video uploads without crashing your server**, the solution is to **stop converting files to Base64** and instead **stream the upload directly to Cloudinary**. This drastically reduces memory usage.  
  
  
u mean i have to use the cloudinary but when i use cloudnry and whn i upload a file and when download this process the much more time  
  
Ahh I see what you mean 👍 — you’re worried about **latency** because Cloudinary is a third-party service, so uploading + downloading videos feels slower than if you just sent files directly.

Let’s break this down:

**⚡ Why Cloudinary felt slower**

* Upload → file travels from client → your server → Cloudinary → then again client has to download from Cloudinary.
* This **double hop** adds delay, especially for large videos.
* Also, Cloudinary optimizes media (thumbnails, adaptive streaming, etc.), which adds processing time.

**✅ Alternatives to Cloudinary**

If you want **faster transfer** (like true peer-to-peer), you have a few options:

**1. Direct Browser-to-Cloud Storage (skip server)**

* Use **presigned URLs** (AWS S3, GCP Storage, Azure Blob, even Cloudinary direct uploads).
* The browser uploads **straight to storage**, bypassing your server.
* Then your server only needs to store the invite code → URL mapping.
* Download happens directly from storage CDN → **faster than relaying through your server**.

👉 This is much faster than the “server middleman” approach you’re doing now.

**2. WebRTC Peer-to-Peer**

* If you truly want **instant file transfer** between users:
  + Use WebRTC dataChannel for direct browser-to-browser transfer.
  + Your server only does **signaling** (exchange connection info).
  + File never touches your server or Cloudinary → **true P2P speed**.
* Downside: If both peers aren’t online at the same time, file can’t be delivered (no persistence).

**3. Hybrid (Best of Both)**

* Use **Cloud Storage (S3, Cloudinary, etc.)** for persistence.
* But **optimize downloads**:
  + For videos → provide streaming (so user doesn’t wait for the whole file).
  + Use CDN (Cloudinary already has this, S3 + CloudFront is super fast).
* For small files (PDFs, images, zips) → WebRTC direct P2P transfer.
* For large videos → fall back to Cloud Storage.

⚡ **Bottom line**:

* If you need **reliable, resumable downloads** → use storage (Cloudinary, S3).
* If you need **fastest transfer but only when both peers are online** → use WebRTC.
* Right now your “server + Cloudinary” flow is **least efficient**   
    
    
  if i want to send 1 min video when i run this project on vs code then it is fast and possible?  
    
  Good question 👍
* Yes — when you run the project **locally in VS Code (localhost)**, sending a **1-minute video** can feel *fast and smooth*, but that speed is a bit misleading:
* **⚡ Why it feels fast on localhost**
* **No internet involved** → File is going from your browser → local server on your PC.
* Network = basically your own machine → super high bandwidth (gigabytes per second).
* No Render, Vercel, or Cloudinary latency.
* So yes, a **1 min video will upload/download quickly** on localhost.
* **🌐 But when deployed online (Render/Vercel/etc.)**
* File has to travel:
* Your browser → remote server (could be in US/EU).
* Server → storage (Cloudinary/S3).
* Other user → download from storage/CDN.
* This adds **upload time** (depends on your internet speed).
* On free tiers (Render/Vercel), servers have **low memory & bandwidth caps** → large files may feel *slow or even crash*.
* **✅ Conclusion**
* **Locally (VS Code)** → 1 min video = fast ✅.
* **Online (deployed)** → 1 min video = slower, depends on:
* your **internet upload speed**,
* your **server’s memory/bandwidth**,
* whether you use **direct-to-cloud (S3/Cloudinary)** or **server as a middleman**.

the main conclusion is for this i have to pay something then i can send the video smoothly  
  
Exactly ✅

That’s the main takeaway:

* **Locally (on your laptop / VS Code)** → no cost, fast, because everything happens on your own machine.
* **Online (deployed)** → if you want smooth, reliable **video uploads/downloads**, especially big files (30 sec, 1 min, 5 min+):
  + Free tiers (Vercel, Render, Cloudinary free plan) usually have **limits** → memory, bandwidth, file size.
  + For smooth experience, you’ll need to move to a **paid plan**:
    - **Cloudinary / AWS S3** → for video storage & delivery.
    - **Vercel / Render** (optional) → just for your web app, not for storing videos.

So yes, **to handle videos smoothly online, some cost is involved** 💳.

👉 But the good news is: if you design it right (upload **directly to Cloudinary/S3** instead of passing through your server), your server costs stay **very low** — you only pay storage & bandwidth for videos.

Do you want me to break down **the cheapest way to support video sharing (with free limits)** before you actually spend money?

Here’s a rough estimate of **how much it might cost** to send / store videos smoothly, using Cloudinary or AWS, and what to watch out for. The actual cost depends on your usage (number of uploads, size of video, how many downloads, etc.).

**🔍 What costs are involved**

If you use Cloudinary or AWS S3, you’ll pay for things like:

* Storage (how many GBs your videos take up)
* Bandwidth / data transfer (downloads, streaming)
* Maybe “transformations” or processing (if you resize video, convert formats, etc.)
* Plan fees / overage fees if you exceed free limits

**💵 Cloudinary cost estimates**

From Cloudinary’s pricing:

* Free plan has **25 credits/month**. Each credit = 1 GB storage OR 1 GB bandwidth OR a certain number of transformations. ([Cloudinary](https://cloudinary.com/pricing?utm_source=chatgpt.com" \o "Cloudinary - Pricing and Plans))
* On the Free plan, max video file size is 100 MB. ([Capterra](https://www.capterra.com/p/135074/Cloudinary/pricing/?utm_source=chatgpt.com))
* If you need larger file size (say videos > 100 MB) or more storage, you’ll probably need the “Plus” plan which cost **$99/month** (or somewhat less if billed annually) for more “credits”. ([The Digital Project Manager](https://thedigitalprojectmanager.com/tools/cloudinary-pricing/?utm_source=chatgpt.com))
* With the Plus plan, you get more storage, more bandwidth, and higher file size limits. ([The Digital Project Manager](https://thedigitalprojectmanager.com/tools/cloudinary-pricing/?utm_source=chatgpt.com))

So: if you upload a 1-minute video say 200-300 MB, store a few of them, and expect people to download/stream, you might:

* Use up a lot of bandwidth ← could push you into paid plan or pay overage.
* Probably need a plan costing ~$100/month or more, depending on how much traffic/downloads you expect.

**💡 AWS S3 cost estimates**

If instead you use AWS S3 for storage + bandwidth:

* Storage on S3 Standard is about **$0.023 per GB per month** for the first ~50 TB. ([Amazon Web Services, Inc.](https://aws.amazon.com/s3/pricing/?utm_source=chatgpt.com))
* Uploads (data going in) are usually free. Downloads (or data transferred out) cost more.
* Also requests (PUT, GET etc.), and transfers outside AWS might cost more. ([Amazon Web Services, Inc.](https://aws.amazon.com/s3/pricing/?utm_source=chatgpt.com))

So for example:

* If you have 10 videos of 200MB each (2 GB total stored), that is ~2 × $0.023 = **$0.046/month** for storage.
* If those videos are downloaded many times, bandwidth cost can add up. If 100 users download those 2 GB videos, that’s 200 GB of transfer, so 200 × (bandwidth cost) = might be a few dollars more (depending on region).

**⚙️ What you should do**

* Estimate how many videos, average size, how many downloads per month.
* Look at Cloudinary’s credit usage vs AWS S3 pricing for your region.
* If traffic is small, free plans might suffice for a while. Once video usage grows, you’ll likely need to move to paid plan.

If you tell me **how many videos per month, average video size, and expected number of downloads**, I can calculate a more precise cost estimate for *your* use-case. Do you want me to do that?

its free version has 512 mb it means i can upload a file upto 512 mb or i can share one file whose size is 50 mb then it means the size left is 462 mb ?  
  
Good question 👍 let’s clear this up:

The **512 MB** on Render Free **is not storage space**.  
It’s the **RAM (memory)** your app can use while running.

**🔍 What this means**

* When you upload a file, the server may load it into memory (RAM) temporarily.
* If the file is **larger than your available memory**, your app can crash.
* So in practice, on **512 MB RAM**, you cannot reliably upload a **512 MB file** — because your app + Node.js + Next.js + buffers already use some memory.
* A safe limit might be **50–100 MB per upload** if you optimize (e.g., stream instead of Buffer).

**❌ What it does *not* mean**

* It does not mean you have a “512 MB bucket” for files where every upload reduces it (like storage).
* After you upload a 50 MB file, you don’t “lose” 50 MB of memory forever. Once the file is sent to Cloudinary (or written to disk and released), the memory is freed.

✅ **Conclusion:**  
On Render Free (512 MB RAM), you can upload files, but the size is limited by memory usage *at upload time*. You don’t have 512 MB total storage, you just have 512 MB RAM to process requests.

If you want to upload/share **bigger videos reliably**, you’ll need:

* **Streaming upload** (so memory isn’t eaten up by the whole file at once).
* Or a **paid plan** (1–2 GB RAM).

👉 Do you want me to rewrite your upload route to use **streaming directly to Cloudinary** so you can push 50–100 MB+ videos even on free 512 MB?

it means i can send as many as i want to send the file of 50-100 mb size to anyonw with the 512mb plan of render  
  
Almost 🙂 let’s be precise:

**✅ What you *can* do on Render Free (512 MB RAM):**

* You can upload/share **multiple files** of 50–100 MB each.
* The **number of files is unlimited**, because Render doesn’t permanently store them — you’re just relaying them to Cloudinary (or another storage).
* After each upload finishes, memory is freed, so you can start another.

**⚠️ The real limits:**

* Each **single upload** must fit into available RAM. On 512 MB free tier:
  + If you use Buffer (whole file in memory) → ~50–100 MB max.
  + If you use **streaming upload** → you can handle bigger files (hundreds of MB) since you never load the entire file into memory.
* There’s also a **time limit per request** (about 15 minutes on Render free plan). Large uploads must complete within that.
* Your **bandwidth** (upload/download speed) depends on Render + Cloudinary network, not your RAM.

✅ So yes — you can send as many 50–100 MB files as you want, **one after another**, on the free plan. You’re not capped at 512 MB total storage, since Render isn’t the storage provider — Cloudinary is.

👉 Do you want me to give you a **safe setup for streaming uploads** so you don’t hit the 50–100 MB ceiling and can push larger videos smoothly?

Web Service PeerLinkS exceeded its memory limit

An instance of your Web Service PeerLinkS exceeded its memory limit, which triggered an automatic restart. While restarting, the instance was temporarily unavailable.

This might have been caused by:

A memory leak in your application

A spike in incoming traffic

An undersized instance type for your use case

Recommended actions

Check your service's logs and metrics to help identify why the instance exceeded its memory limit.

If you find a memory leak, patch the leak and redeploy your service.

If you experienced a traffic spike, consider scaling your service to help handle future spikes.

Otherwise, your service might require more memory for your use case. To increase memory, upgrade your instance type.

If you take no action, your service might exceed its memory limit again, resulting in additional interruptions.

If you can't resolve this issue, please reach out to support@render.com.

Best,

The Render team

You can change notification settings for this service, along with default notification settings for your account/team.

this mail comes when i try to upload a video of size 150 mb  
  
  
  
Ahh, that explains it 👍

Here’s why you got that Render email when uploading a **150 MB video**:

**🔍 What happened**

* Your current upload code uses Buffer (or arrayBuffer() → Buffer) → this loads the **entire file into server memory** before sending it to Cloudinary.
* On Render Free, you only have **512 MB RAM** total.
  + Node.js + Next.js itself already eats ~200–250 MB.
  + A 150 MB video in memory can expand to more (Base64 adds ~33% overhead).
  + That pushes your app **over 512 MB**, so Render automatically **kills and restarts** your instance.

That’s exactly why you got the **“exceeded its memory limit”** email.

**✅ The solution**

You need to switch to **streaming uploads** instead of buffering.

* With streaming, your server never holds the entire 150 MB file in memory.
* The file gets piped directly from the request → to Cloudinary (or S3).
* That means even huge files (hundreds of MB) can upload on the 512 MB free plan, without crashing.

when i deploy this nextjs project on vercel the project not work why  
  
  
Got it 👍 this is a very common issue when deploying **Next.js + file uploads** to **Vercel**.

**🔍 Why your project doesn’t work on Vercel**

1. **Vercel is serverless**
   * Each API route runs in a short-lived serverless function.
   * You **can’t write to local disk (fs)** — /uploads or fileDB.json won’t persist.
   * Every request is stateless, so your uploads folder disappears after the function finishes.
2. **Large file uploads are not supported**
   * Vercel functions have strict **body size limits (4.5 MB)**.
   * That means your 50 MB or 150 MB video upload will always fail.
3. **Memory limits**
   * Serverless functions have ~128–1024 MB RAM depending on plan.
   * But for free tier, handling large buffers/videos is impossible.

**✅ What this means for you**

* Your current project design (upload → save file locally → serve) will **never work on Vercel**.
* Even with Cloudinary, if you use Buffer, it will break because of Vercel’s **4.5 MB body limit**.

**🚀 The correct way to use Vercel for video/file sharing**

* **Don’t upload files to Vercel functions at all.**
* Instead, upload **directly from the client (browser) to Cloudinary/S3**.
  + Your Next.js API only issues a **signed upload URL** (very tiny request, works fine).
  + The browser then sends the file straight to Cloudinary, bypassing Vercel’s limits.
* When someone wants to download, you serve them the **Cloudinary URL** or redirect to it.

This is the only way to make file/video sharing work reliably on Vercel.

👉 So to answer simply:  
Your project doesn’t work on Vercel because **Vercel blocks large uploads and doesn’t support local file storage**.

Do you want me to show you how to **rewrite your upload flow for Vercel** (client → Cloudinary direct upload with signed URL)? That way it will work even for large videos.

There is also a limit on vs code to send the file upto what size

Great question 👍 let’s clear this up:

When you run your project **locally in VS Code**, there is **no fixed limit** like Vercel or Render impose.

**🚀 On VS Code (local machine)**

* Your app runs directly on your laptop.
* File size limit = whatever your system can handle.
* Constraints are:
  + **Your RAM** → if you buffer the entire file, you need enough memory.
  + **Your disk space** → if you’re saving locally.
  + **Your internet speed** → for uploading to Cloudinary/S3.

For example:

* If your laptop has **8 GB RAM**, you could upload even a **1 GB file** locally (though it might be slow).
* There’s no artificial **4.5 MB or 512 MB cap** like on Vercel/Render Free.

**⚠️ But remember**

* If you keep using Buffer to hold the full file in memory, the bigger the file → the more RAM it consumes.
* That’s why a **streaming upload** is better — it works for files of *any size*, limited only by your internet.

✅ **Conclusion:**  
On VS Code (local), you can upload **as big a file as your machine + internet allow**. There’s no strict “max size”. But for safety and scalability, switch to **streaming uploads** so you’re not limited by RAM.

👉 Do you want me to give you a **side-by-side comparison** of how file size limits differ between **VS Code (local), Render Free, and Vercel**? That way you’ll see exactly where things break.