Server Uploads with Vercel Blob

Vercel Blob is available on all plans

Those with the owner, member, developer role can access this feature

In this guide, you'll learn how to do the following:

- Use the Vercel dashboard to create a Blob store connected to a project
- Upload a file using the Blob SDK from the server

Vercel has a 4.5 MB request body size limit on Vercel Functions. If you need to upload larger files, use client uploads.

Prerequisites

Vercel Blob works with any frontend framework. First, install the package:

pnpm yarn npm bun

```
pnpm i @vercel/blob
```

1 Create a Blob store

Navigate to the Project you'd like to add the blob store to. Select the **Storage** tab, then select the **Connect Database** button.

Under the **Create New** tab, select **Blob** and then the **Continue** button.

Use the name "Images" and select **Create a new Blob store**. Select the environments where you would like the read-write token to be included. You can also update the prefix of the Environment Variable in Advanced Options

Once created, you are taken to the Vercel Blob store page.

2 **Prepare your local project**

Since you created the Blob store in a project, we automatically created and added the following Environment Variable to the project for you.

- `BLOB_READ_WRITE_TOKEN`

To use this Environment Variable locally, we recommend pulling it with the Vercel CLI:

```
vercel env pull
```

Server uploads are perfectly fine as long as you do not need to upload files larger than 4.5 MB on Vercel. If you need to upload larger files, consider using client uploads.

Upload a file using Server Actions

The following example shows how to use a Server Action with Next.js App Router to upload a file to Vercel Blob.

app/components/form.tsx

```tsx
import { put } from '@vercel/blob';
import { revalidatePath } from 'next/cache';

export async function Form() {
  async function uploadImage(formData: FormData) {
    'use server';
    const imageFile = formData.get('image') as File;
    const blob = await put(imageFile.name, imageFile, {
      access: 'public',
      addRandomSuffix: true,
    });
    revalidatePath('/');
    return blob;
  }

  return (
    <form action={uploadImage}>
      <label htmlFor="image">Image</label>
      <input type="file" id="image" name="image" required />
      <button>Upload</button>
    </form>
  );
}
```

Then, add the hostname to your `next.config.js` file including the store id from the dashboard:

next.config.js

```js
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'my-blob-store.public.blob.vercel-storage.com',
        port: '',
      },
    ],
  },
};

module.exports = nextConfig;
```

This will allow you to use `next/image` to display images from your Vercel Blob store.

app/components/images.tsx

```tsx
import { list } from '@vercel/blob';
import Image from 'next/image';

export async function Images() {
  async function allImages() {
    const blobs = await list();
    return blobs;
  }
  const images = await allImages();

  return (
    <section>
      {images.blobs.map((image) => (
        <Image
          priority
          key={image.pathname}
          src={image.url}
          alt="Image"
          width={200}
          height={200}
        />
      ))}
    </section>
  );
}
```

Read more about Server Actions and App Router on the Next.js documentation.

Upload a file using a server upload page and route

You can upload files to Vercel Blob using Route Handlers/API Routes. The following example shows how to upload a file to Vercel Blob using a server upload page and route.

1 Create a server upload page

This page will upload files to your server. The files will then be sent to Vercel Blob.

Next.js (/app) Next.js (/pages)

src/app/avatar/upload/page.tsx TypeScript

```
'use client';

import type { PutBlobResult } from '@vercel/blob';
import { useState, useRef } from 'react';

export default function AvatarUploadPage() {
  const inputFileRef = useRef<HTMLInputElement>(null);
  const [blob, setBlob] = useState<PutBlobResult | null>(null);
  return (
    <>
      <h1>Upload Your Avatar</h1>

      <form
        onSubmit={async (event) => {
          event.preventDefault();

          if (!inputFileRef.current?.files) {
            throw new Error('No file selected');
          }

          const file = inputFileRef.current.files[0];

          const response = await fetch(
            `/api/avatar/upload?filename=${file.name}`,
            {
              method: 'POST',
              body: file,
            },
          );

          const newBlob = (await response.json()) as PutBlobResult;

          setBlob(newBlob);
        }}
      >
        <input name="file" ref={inputFileRef} type="file" required />
        <button type="submit">Upload</button>
      </form>
      {blob && (
        <div>
          Blob url: <a href={blob.url}>{blob.url}</a>
        </div>
      )}
    </>
  );
}
```

2 Create a server upload route

This route forwards the file to Vercel Blob and returns the URL of the uploaded file to the browser.

Next.js (/app) Next.js (/pages)

src/app/api/avatar/upload/route.ts TypeScript

```ts
import { put } from '@vercel/blob';
import { NextResponse } from 'next/server';

export async function POST(request: Request): Promise<NextResponse> {
  const { searchParams } = new URL(request.url);
  const filename = searchParams.get('filename');

  const blob = await put(filename, request.body, {
    access: 'public',
    addRandomSuffix: true,
  });

  return NextResponse.json(blob);
}
```

Testing your page

1 Run your application locally

Run your application locally and visit `/avatar/upload` to upload the file to your store. The browser will display the unique URL created for the file.

When your local website is served on `http://localhost:3000`, then the `onUploadCompleted` step won't succeed as Vercel Blob cannot contact your localhost. Instead, we recommend you run your local application through a tunneling service like ngrok, so you can experience the full Vercel Blob development flow locally.

2 Review the Blob object metadata

- Go to the Vercel Project where you created the store
- Select the **Storage** tab and select your new store

  - Paste the blob object URL returned in the previous step in the **Blob URL** input box in the **Browser** section and select **Lookup**
  - The following blob object metadata will be displayed: file name, path, size, uploaded date, content type and HTTP headers
  - You also have the option to download and delete the file from this page

You have successfully uploaded an object to your Vercel Blob store and are able to review it's metadata, download, and delete it from your Vercel Storage Dashboard.

Next steps

- Learn how to use the methods available with the `@vercel/blob` package

Last updated on March 4, 2025

Next
Account Management

Was this helpful?