Building a native file upload handler for Caddy v2

2022-03-01

How I built caddyv2-upload, a native file upload handler for Caddy v2 written in Go — the problem it solves, and how it works.

When Caddy v2 was released in 2020, it introduced a new concept of handlers that use Go's native module features.

In 2022 one of our customers needed to upload files, and at that time none of the available web servers could handle this natively without an extra program. So I decided to create a new handler for Caddy v2 — caddyv2-upload was born.

I was not aware of any other tool which can handle file uploads natively in just one binary which can be used in Kubernetes or OpenShift, or I have not searched hard enough :grin:. I also wanted to learn go, so I decided to create a new handler for the go based webserver caddyserver.

In May 2022 dufs, a Rust-based server, was created — so that might be another option worth considering. :shrug:

Any how I wanted to learn go in the pre AI Area and at that time I looked into the docs of go modules, into the source of caddy servers own handlers and created step by step a new handler.

Luckily there was already podman and buildah, so I was able to create container images to run and test locally.

Building something real turned out to be the most effective way to learn Go. Diving into Caddy's own handler source code, understanding the module system, and iterating with containers made the concepts stick far better than any tutorial would have.

A minimal Caddyfile configuration looks like this:

```
{
    order upload before file_server
}

:2015 {
    root .

    file_server browse

    @mypost method POST
```

```
    upload @mypost {
        dest_dir /var/uploads
        max_filesize 4MB
    }
}
```

Build with xcaddy:

```
xcaddy build --with github.com/git001/caddyv2-upload
```

Today caddyv2-upload (v0.20.0) supports file uploads via HTTP multipart form, configurable destination directories, UUID-based subdirectory creation per upload, fixed filenames, pass-through mode to continue to the next Caddy handler after upload, configurable file size limits, an HTTP notification callback after successful uploads, and a `/health` endpoint. It integrates with Caddy's variable system and works in Kubernetes and OpenShift environments. Authentication is handled by Caddy's own security layer rather than the module itself.