# File Upload

```
curl -v -X POST http://localhost:18080/uploadfile -F \
"InputFile=@/home/inbe1e-dl2609cs/Documents/Datasheet_Activities/tps3851-q1_wachdog.pdf"
```

Outputs this if the server is already running

```
Note: Unnecessary use of -X or --request, POST is already inferred.
* Host localhost:18080 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:18080...
* connect to ::1 port 18080 from ::1 port 58842 failed: Connection refused
*   Trying 127.0.0.1:18080...
* Connected to localhost (127.0.0.1) port 18080
> POST /uploadfile HTTP/1.1
> Host: localhost:18080
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Length: 1886627
> Content-Type: multipart/form-data; boundary=------------------------cN44k8HYPDLVtRnNrck91H
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Content-Length: 0
< Server: Crow/master
< Date: Sun, 20 Jul 2025 17:03:57 GMT
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
```

---

```
#include "crow.h"

int main()
{
    crow::SimpleApp app;
    CROW_ROUTE(app, "/uploadfile")
      .methods(crow::HTTPMethod::Post)([](const crow::request& req) {
          crow::multipart::message_view file_message(req);
          for (const auto& part : file_message.part_map)
          {
              const auto& part_name = part.first;
              const auto& part_value = part.second;
              CROW_LOG_DEBUG << "Part: " << part_name;
              if ("InputFile" == part_name)
              {
                  // Extract the file name
                  auto headers_it = part_value.headers.find("Content-
Disposition");
                  if (headers_it == part_value.headers.end())
                  {
                      CROW_LOG_ERROR << "No Content-Disposition found";
                      return crow::response(400);
                  }
                  auto params_it = headers_it-
>second.params.find("filename");
                  if (params_it == headers_it->second.params.end())
                  {
                      CROW_LOG_ERROR << "Part with name \"InputFile\" should
have a file";
                      return crow::response(400);
                  }
                  const std::string outfile_name{params_it->second};

                  for (const auto& part_header : part_value.headers)
                  {
                      const auto& part_header_name = part_header.first;
                      const auto& part_header_val = part_header.second;
                      CROW_LOG_DEBUG << "Header: " << part_header_name <<
'=' << part_header_val.value;
                      for (const auto& param : part_header_val.params)
                      {
                          const auto& param_key = param.first;
                          const auto& param_val = param.second;
                          CROW_LOG_DEBUG << " Param: " << param_key << ','
<< param_val;
                      }
                  }
                  // Create a new file with the extracted file name and write file contents to it
                  std::ofstream out_file(outfile_name);
                  if (!out_file)
                  {
                      CROW_LOG_ERROR << " Write to file failed\n";
                      continue;
                  }
                  out_file << part_value.body;
                  out_file.close();
                  CROW_LOG_INFO << " Contents written to " << outfile_name << '\n';
              }
              else
              {
                  CROW_LOG_DEBUG << " Value: " << part_value.body << '\n';
              }
          }
          return crow::response(200);
      });

    // enables all log
    app.loglevel(crow::LogLevel::Debug);

    app.port(18080)
      .multithreaded()
      .run();

    return 0;
}
```

---

```
. my_crow_project/
├── file_upload_project/
│   ├── build/
│   ├── CMakeLists.txt
│   └── main.cpp
└── libs/
    ├── asio/
    └── crow/
```

```
cmake_minimum_required(VERSION 3.15)
project(MyCrowProject)
```

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Threads REQUIRED)

# Add Crow include directory
include_directories(${CMAKE_SOURCE_DIR}/../libs/crow/include)

# Add ASIO include directory (standalone)
include_directories(${CMAKE_SOURCE_DIR}/../libs/asio/asio/include)

# Define ASIO_STANDALONE to use ASIO without Boost
add_definitions(-DASIO_STANDALONE)

# Optional: Disable ASIO's deprecated features
add_definitions(-DASIO_NO_DEPRECATED)

# Create executable
add_executable(file_upload_project main.cpp)

# Link threads
target_link_libraries(file_upload_project Threads::Threads)

if(WIN32)
    target_link_libraries(file_upload_project ws2_32 wsock32)
endif()

if(UNIX)
    target_link_libraries(file_upload_project pthread)
endif()
```

---

> Install aarch64 gnu tool chain

```
sudo apt-get update
sudo apt-get install g++-aarch64-linux-gnu gcc-aarch64-linux-gnu
```

> Add tool-chain file `aarch64-toolchain.cmake` to project `root` where `CMakeLists.txt` exist

```
. my_crow_project/
├── file_upload_project/
|   ├── build/
|   ├── CMakeLists.txt
|   ├── aarch64-toolchain.cmake
|   └── main.cpp
└── libs/
    ├── asio/
    └── crow/
```

> The contents of `aarch64-toolchain.cmake` looks like this:

```cmake
# aarch64-toolchain.cmake
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

# Optional: set the sysroot if needed
# set(CMAKE_SYSROOT /usr/aarch64-linux-gnu)

# Optional: set search paths for libraries and includes
# set(CMAKE_FIND_ROOT_PATH /usr/aarch64-linux-gnu)
```

> Configure and build

```
cmake -B build-aarch64 -DCMAKE_TOOLCHAIN_FILE=./aarch64-toolchain.cmake
cmake --build build-aarch64
```

---

To run and see the aarch64 binary in x86, use docker VM

```
.
├── Dockerfile.dev
├── file_upload_server
│   ├── aarch64-toolchain.cmake
│   ├── build-aarch64
│   ├── CMakeLists.txt
│   └── main.cpp
└── libs
    ├── asio
    └── crow

6 directories, 4 files
```

In `Dockerfile.dev`

```dockerfile
# Development environment for ARM64 building
FROM --platform=linux/arm64 debian:bookworm-slim
```

```
# Set environment to avoid interactive prompts
ENV DEBIAN_FRONTEND=noninteractive

# Set the working directory
WORKDIR /workspace

# Install development tools and dependencies
RUN apt-get update && \
    apt-get install -y \
        cmake \
        make \
        g++ \
        gcc \
        build-essential \
        libstdc++6 \
        libc6 \
        libgcc-s1 \
        ca-certificates \
        git \
        pkg-config && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Expose port if needed
EXPOSE 8080

# Default command to keep container running
CMD ["/bin/bash"]
```

Run the below command from where the Dockerfile is present

```
docker buildx build --platform=linux/arm64 -f ${PWD}/Dockerfile.dev -t
arm64-dev:latest --load ${PWD}
```

```
docker run --platform=linux/arm64 -it -p 8081:8080 -v ${PWD}:/workspace
arm64-dev:latest
```

```
# to remove the container before deleting the image
docker container prune -f
```

```
docker rmi -f arm64-dev:latest
```