

Boost
Boost.beast
Boost.asio

Antonio Espín Herranz

Boost

- **Boost** es una **colección de librerías de C++** que cubren múltiples áreas de desarrollo.
 - <https://www.boost.org/library/latest/beast/>
 - Smart pointers → `boost::shared_ptr`
 - Expresiones regulares → `boost::regex`
 - Manejo de archivos → `boost::filesystem`
 - **Protocolos HTTP y WebSockets** → **`boost::beast`**
 - **`boost::beast` depende de `boost::asio`** → manejo de operaciones de E/S, peticiones asíncronas (red, sockets)
 - https://www.boost.org/doc/libs/latest/doc/html/boost_asio.html

Boost.Beast

- Nos proporciona manejo de peticiones **Http** y **WebSockets**
- Para **instalar**:
 - **vcpkg install boost-beast**
- **Integrar** en Visual Studio:
 - **vcpkg integrate install**
- Listar librerías:
 - **vcpkg list**
- Comprobar si la tenemos instalada:
 - **vcpkg list | findstr boost-beast**

Características de la librería

- **Boost.Beast** es una biblioteca **header-only** (solo incluye cabeceras, no requiere compilación previa) que proporciona componentes de bajo nivel para manejar protocolos de red como **HTTP/1.1** y **WebSocket**,
- Con soporte para operaciones **síncronas** y **asíncronas**

Características de la librería II

- Basada en **Boost.Asio**
 - Usa el modelo **asincrónico de Boost.Asio**, lo que permite construir aplicaciones altamente concurrentes.
 - Compatible con `io_context`, `executors`, y operaciones compuestas.
- Protocolos soportados
 - **HTTP/1.1**: Lectura, escritura, serialización y análisis de mensajes HTTP.
 - **WebSocket**: Comunicación bidireccional en tiempo real, incluyendo control frames y compresión.

Características III

- **Abstracciones de flujo (Streams)**
 - `basic_stream`, `tcp_stream`, `ssl_stream`: para manejar conexiones TCP/IP, con o sin cifrado.
- Soporte para **SSL/TLS** mediante integración con **OpenSSL**.
- Gestión de **buffers**
 - `flat_buffer`, `multi_buffer`, `static_buffer`: para optimizar el manejo de datos en red.
- **Flexibilidad**
 - El **desarrollador controla** aspectos como el manejo de **buffers**, **hilos**, y políticas de tasa de transferencia.
 - Ideal para construir tanto **clientes** como **servidores**, gracias a su diseño simétrico.
- **Extensibilidad**
 - Sirve como base para construir bibliotecas de red más complejas.
 - Bien adaptada para integrarse en arquitecturas de microservicios o sistemas distribuidos.

Requisitos

- Página oficial: <https://www.boost.org/library/latest/beast/>
- Wiki: <https://deepwiki.com/boostorg/beast>
- Necesitamos versiones **\geq C++11**
- **Boost.Asio** y otras partes de Boost.
- **OpenSSL** si se desea soporte para conexiones seguras.
- Compatible con **Visual Studio 2017+**, **CMake** \geq 3.5.1, para construir ejemplos

Desglose de la librería Boost.Beast

Núcleo de Boost.Beast

- **Buffers y Streams:** Beast proporciona sus propios tipos de buffer (`flat_buffer`) y abstrae los streams para facilitar la lectura/escritura de datos en conexiones TCP.
- **Integración con Boost.Asio:** Toda la funcionalidad de Beast se basa en Asio, lo que permite usar operaciones sincrónicas y asincrónicas con coroutines, callbacks o `async/await`.

Manejo de HTTP

- **HTTP Messages**

- `http::request<T>` y `http::response<T>`: Representan mensajes HTTP con cuerpo de tipo T (como `string_body`, `file_body`, etc.).
- `http::fields`: Encapsula los encabezados HTTP.

- **HTTP Operations**

- `http::read` / `http::async_read`: Leer peticiones o respuestas desde un stream.
- `http::write` / `http::async_write`: Enviar peticiones o respuestas por el stream.

- **HTTP Server y Client**

- Beast permite construir **servidores HTTP** y **clientes HTTP** usando sockets **TCP** o **SSL**.
- Soporta **HTTP/1.1** (no **HTTP/2** aún de forma nativa).

Manejo de WebSockets

- **WebSocket Stream**

- `websocket::stream<T>`: Abstrae una conexión WebSocket sobre un stream TCP o SSL.

- **Operaciones WebSocket**

- `websocket::handshake / async_handshake`: Realiza el handshake inicial para establecer la conexión.
- `websocket::read / async_read`: Recibe mensajes WebSocket.
- `websocket::write / async_write`: Envía mensajes WebSocket.
- `websocket::close`: Cierra la conexión de forma ordenada.

Manejo de WebSockets 2

- **Soporte de Frames**

- Permite enviar y recibir **frames de texto o binarios**, con control sobre fragmentación y flags.

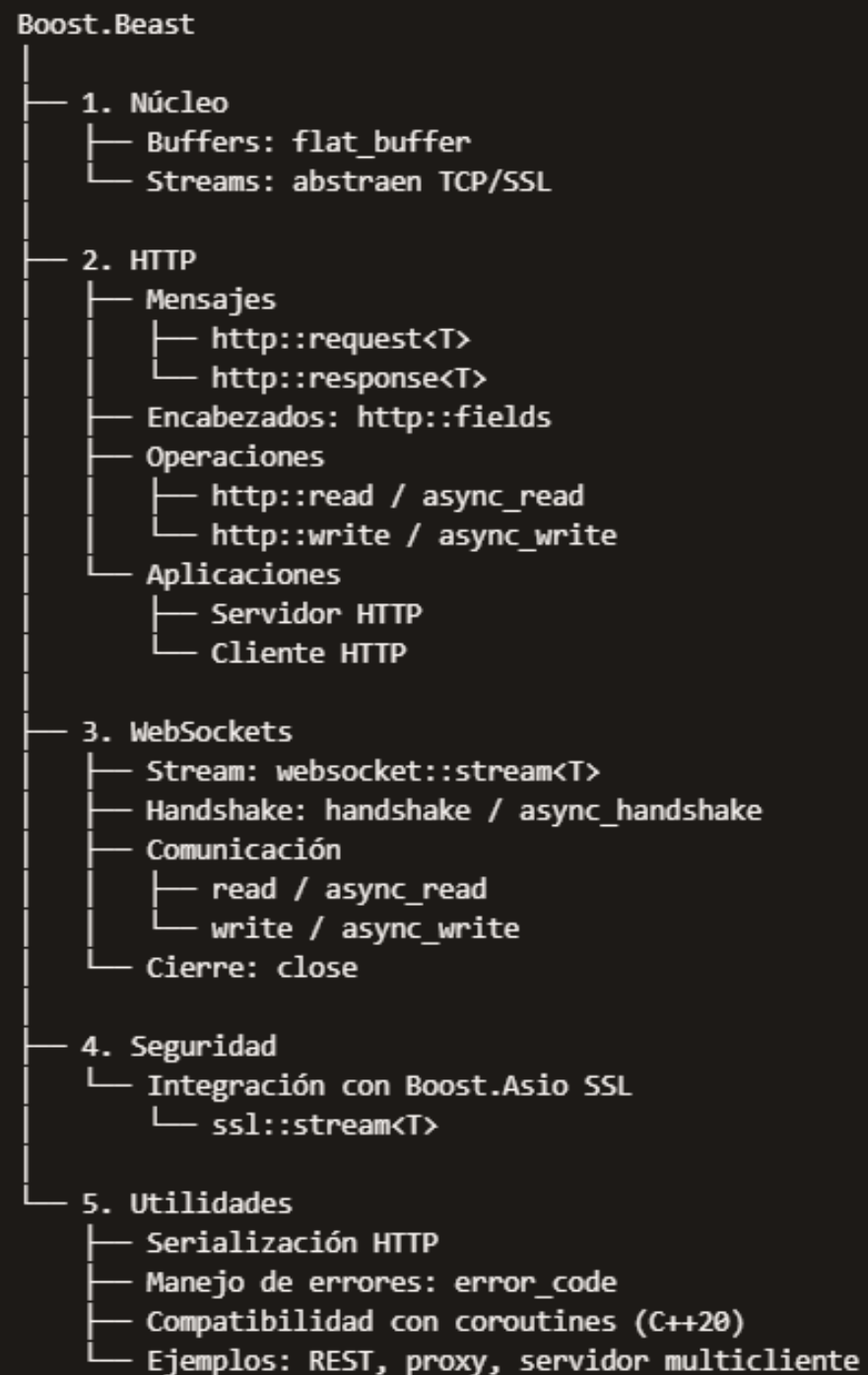
Seguridad y SSL

- Beast no gestiona SSL directamente, pero se integra perfectamente con Boost.Asio SSL.
- Puedes envolver los streams con `boost::asio::ssl::stream` para manejar conexiones seguras.

Utilidades y Extras

- Serialización y parsing de mensajes HTTP.
- Control de errores mediante `boost::system::error_code`.
- Compatibilidad con coroutines (`co_await`, `co_spawn`) en C++20.
- Ejemplos y patrones para servidores multicliente, proxies, y clientes REST.

Organización interna



Tipos de aplicaciones Http con Boost.Beast

- Se pueden implementar una **gran variedad de aplicaciones** de red.
 - Manejando **protocolos** como **Http** y **WebSockets**.
- **1 - Cliente HTTP**
 - Realiza **peticiones** a **servidores** externos (GET, POST, PUT, DELETE...).
 - Ideal para **consumir** APIs **REST** desde **C++**.
 - Puedes manejar encabezados, cuerpos JSON, autenticación, etc.
 - *Ejemplo: Un cliente que consulta la API de OpenWeather y muestra el clima en Madrid.*

Tipos de aplicaciones Http con Boost.Beast

- **2 - Proxy HTTP**

- Recibe peticiones de clientes y las redirige a otros servidores.
- Puede modificar encabezados, filtrar contenido o registrar tráfico.
- *Ejemplo: Un proxy que añade autenticación a peticiones antes de reenviarlas a un backend.*

- **3 - API RESTful**

- **Servidor** que expone **endpoints** como /usuarios, /productos, etc.
- Maneja rutas, métodos HTTP y respuestas en JSON.
- Se puede integrar con bases de datos y lógica de negocio.
- *Ejemplo: Una API para gestionar inventario desde una app móvil.*

Tipos de aplicaciones Http con Boost.Beast

- **4 - Microservicio HTTP**

- Aplicación ligera que realiza una tarea específica (ej. validación, cálculo, logging).
- Se comunica con otros servicios vía HTTP o WebSocket.
- Ideal para arquitecturas distribuidas.
- *Ejemplo: Un microservicio que calcula precios con IVA y responde en milisegundos.*

- **5 - Servidor de archivos estáticos**

- Sirve HTML, CSS, JS, imágenes y otros recursos desde disco.
- Útil para alojar páginas web o documentación técnica.
- *Ejemplo: Un servidor que entrega una SPA (Single Page Application) compilada en React.*

Tipos de aplicaciones Http con Boost.Beast

- **6 - Servidor de streaming HTTP**
 - Envía datos en tiempo real usando chunked encoding o SSE (Server-Sent Events).
 - Útil para dashboards, logs en vivo o feeds de eventos.
 - *Ejemplo: Un servidor que transmite métricas de sensores cada segundo.*
- **7 - Servidor de autenticación**
 - Maneja login, tokens JWT, sesiones y autorización.
 - Puede integrarse con OAuth2, LDAP o bases de datos.
 - *Ejemplo: Un backend que valida credenciales y emite tokens para apps cliente.*
- ***WebSocket estaría más enfocado para la comunicación en tiempo real.***