Mercure-Hub: Getting started

How to start using Mercure-Hub

> 👋**Welcome on Stackhero's documentation!**
> Stackhero provides Mercure-Hub instances that are ready for production in just 2 minutes!
> Including TLS encryption (aka HTTPS), customizable domain name, unlimited messages and size, backups and updates in just a click.
> Try our managed Mercure-Hub cloud in just 2 minutes

What is Mercure-Hub

Let's imagine you have a website selling books and that you want to show to your customers how many books you have in your stock **in realtime**.

Pushing data from the back-end to the front-end can be really difficult.

Mercure will simplify that and in few minutes you will be able to push data to your customers' browsers!

And the best part, it's compatible with any language as it uses HTTP protocol!

How Mercure-Hub works

A customer is looking at a book. This book has the ID 1.

On the front-end, we will subscribe to the topic `/books/1` on Mercure-Hub.
To do that, we will use the SSE (Server-Sent Events) API, which is a native HTML 5 API.
It's around 10 JavaScript lines, **without using any external library**: it's super easy and terribly efficient!

On the back-end side, when a book is bought by someone, we will send an HTTP request to Mercure-Hub to update the stock.

Imagine we have 7 books with ID 1.
A user buy one of this book. We now have 6 books.

Our back-end will send `{ stockCount: 6 }` to the topic `/books/1` on Mercure-Hub.
Every user that is looking at the book ID 1 will receive instantly the new stock count!
All of this with just an HTTP request from the back-end and 10 lines of code on the front-end.

You can use this principle to push data from the server to the client, between clients or even between servers.

Getting started with Mercure-Hub

On the client side (your front-end), you just have to put some simple JavaScript code. This code will use the HTML5 SSE API.

> Note: in this example, we don't authenticate the subscriber.
> You have to allow anonymous subscribers on Stackhero dashboard to make this example works.

```
const endpoint = 'XXXXXX.stackhero-network.com';

const url = new URL('https://' + endpoint + '/.well-known/mercure');

// Add topic to listen to
url.searchParams.append('topic', `https://${endpoint}` + '/users/1234');
url.searchParams.append('topic', `https://${endpoint}` + '/books/1');

const eventSource = new EventSource(url);

// The callback will be called every time an update is published
eventSource.onmessage = e => console.log(e);
```

On the server side (your back-end), when you want to dispatch an update, you just have to run a POST request to the Mercure-Hub API.

Here is an example with Node.js, JsonWebToken and Request libraries:

```
const jwt = require('jsonwebtoken');
const request = require('request');

const endpoint = 'XXXXXX.stackhero-network.com';
const publisherJwtKey = '<your publisher JWT key>';

// Defining the data we want to dispatch
const data = {
  // The topic where the data will be pushed
  topic: `https://${endpoint}` + '/books/1',

  // The data that will send to the topic
  data: JSON.stringify({
    available: false,
    date: new Date()
  })
};


// Generating the bearer
```

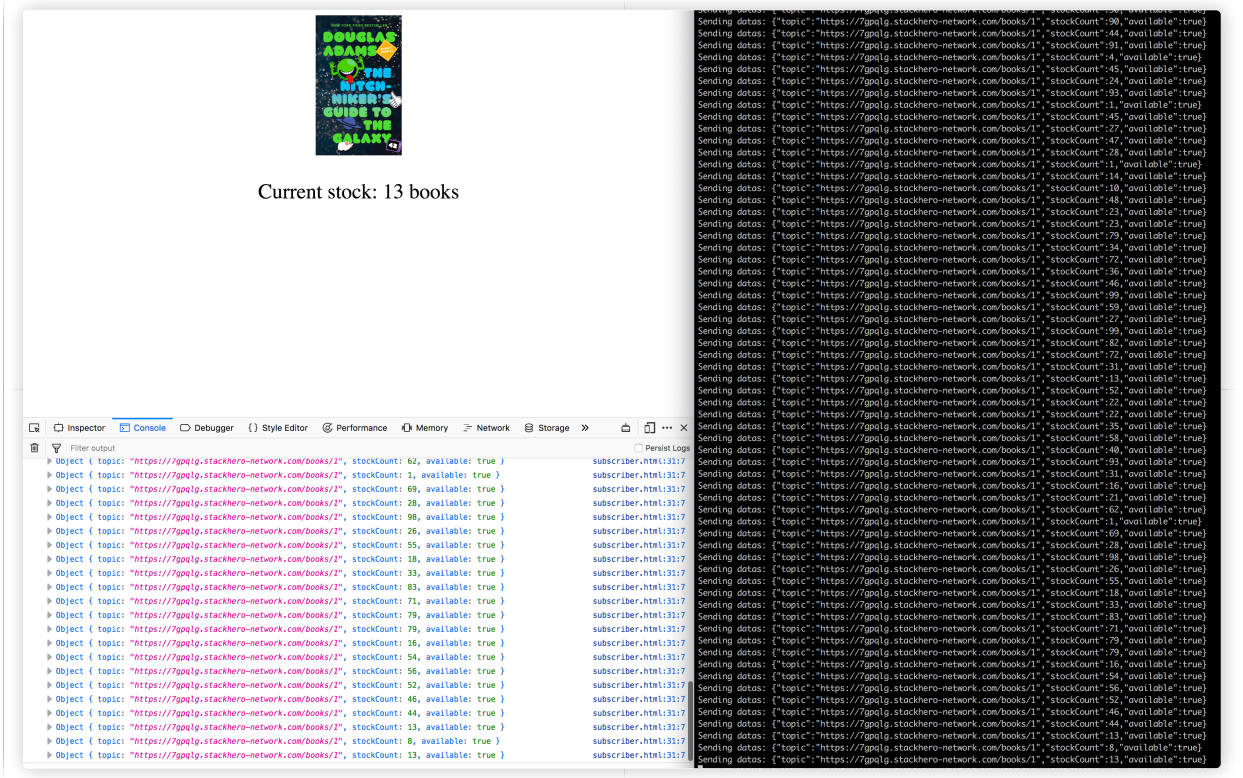
```
const bearer = jwt.sign(
  { mercure: { publish: [ data.topic ] } },
  publisherJwtKey,
  {
    expiresIn: 60, // Bearer expiring in one minute
    noTimestamp: true // Do not add "issued at" information to avoid error "Token
used before issued"
  }
);


// Sending the data to Mercure-Hub that will dispatch them to clients
request.post(
  {
    url: `https://${endpoint}/.well-known/mercure`,
    auth: { bearer },
    form: data
  },
  (err, res) => err ? console.error(err) : console.log(res)
);
```

That's it! You now have a subscriber (on the front-end side) and a publisher (on the back-end side)!

Mercure-Hub client code example

If you want to test Mercure-Hub, you can use our code examples available on https://github.com/stackhero-io/mercureHubGettingStarted.

They are working examples with a simple front-end page and a Node.js back-end server. Perfect to test and better understand Mercure.



Simple example of a back-end and frond-end communication with Mercure-Hub

Authentication of subscribers

In our previous examples, we didn't authenticate subscribers and had to allow "anonymous subscribers" in Stackhero dashboard.

To authenticate subscribers, we will have to generate a JWS (JSON Web Signature) based on the "Subscriber JWT key" defined in Stackhero dashboard.

This JWS will then be sent using the browser's cookies or the header `authorization`.

As SSE (Server-Sent Events) doesn't support headers definition, we have to use cookies. Unfortunately, using cookies adds a big limitation: our Mercure-Hub server and our client have to use the same domain (or subdomain).

To use SSE and different domains, we can use an EventSource polyfill that allows us to define headers. In our example, we will use this one: https://github.com/Yaffle/EventSource.

First you have to generate a JWS for your client. It has to be done on your server side.
You will get an example in `backend/subscriberJwsGenerator.js`. Just fill your subscriber JWT and run the script with `node subscriberJwsGenerator.js`.

Then, on the front-end side, in the file `frontend/subscriberWithAuthorization.html`, fill your endpoint and the JWS you just got.
Open the file in your browser and tada, your Mercure-Hub works with authentication!

> Don't forget to uncheck the "Allow anonymous subscribers" in Stackhero dashboard!