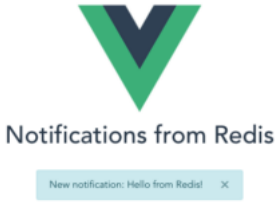
How to Create Notification Services with Redis, Websockets, and Vue.js

See how easy it is to create a small notification service using Redis Pub/Sub to send messages to a web application, with Vue.js, Node.js, and WebSockets.



August 19, 2020

by Tugdual Grall
Technical Marketing Manager

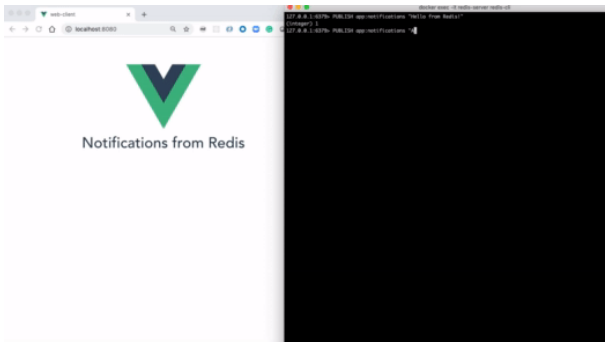
Tags: notifications, Redis Streams

← Back to Blogs

common to get real-time notifications when navigating in a lication. Notifications could come from a chat bot, an alerting or be triggered by an event that the app pushes to one or more natever the source of the notifications, developers are gly using Redis to create notification services.

n applications powered by a microservices architecture, Redis ised as a simple cache and as a primary database. But it is also communication layer between services using a persistent messaging layer powered by Redis Streams, a lightweight eventing system using its well-known Pub/Sub (Publish/Subscribe) commands.

In this blog post, we'll show you how easy it is to create a small notification service using Redis Pub/Sub to send messages to a web application, developed with Vue.js, Node.js, and WebSockets.



Here's how the notifications work. For a higher-resolution version, click on this gif to see the video.

Prerequisites

This demo service uses:

- Docker
- Redis 5.0.x or later (this demo uses the Redis Docker container)
- Node.js 10.x with Node package manager (npm)
- nodemon, a simple tool that automatically restarts your application during development
- Vue CLI

Starting Redis server

If you do not have already a Redis instance running, you can start it using Docker; in a terminal, run this command:

```
> docker run -it --rm --name redis-server -p 6379:6379 redis
```

Redis should now be up and running and ready to accept connections.

Creating the WebSocket server with Node.js

To configure the project with the proper structure, open a terminal and enter the following commands:

```
> mkdir notifications

> cd notifications

> mkdir notif-server

> cd notif-server
```

001-create-project.sh hosted with ❤ by **GitHub** **view raw**

Create a new Node.js project using npm *(the -y parameter will set all values to the default one)*:

```
> npm init -y

> npm install ws redis
```

002-npm-init.sh hosted with ❤ by **GitHub** **view raw**

The final command above adds the WebSocket and Redis dependencies to your project. You are now ready to write some code!

Writing the WebSocket server

Open your favorite code editor for Node.js (I use Visual Studio Code) and simply enter the code "`code .`" to open the current directory. In your editor, create a new file called `server.js`.

```
const WebSocket = require('ws');
const redis = require('redis');

// Configuration: adapt to your environment
const REDIS_SERVER = "redis://localhost:6379";
const WEB_SOCKET_PORT = 3000;

// Connect to Redis and subscribe to "app:notifications"
var redisClient = redis.createClient(REDIS_SERVER);
redisClient.subscribe('app:notifications');

// Create & Start the WebSocket server
const server = new WebSocket.Server({ port : WEB_SOCKET_

// Register event for client connection
server.on('connection', function connection(ws) {

  // broadcast on web socket when receving a Redis PUB/S
  redisClient.on('message', function(channel, message){
    console.log(message);
    ws.send(message);
  })

});

console.log("WebSocket server started at ws://locahost:"
```

003-write-websocket-server.js hosted with ❤ by **GitHub** **view raw**

This simple Node.js program is limited to the demonstration and focuses on:

- Connecting to Redis (line 9)
- Subscribing to the messages from the "`app:notifications`" channel (line 10)
- Starting a WebSocket server (line 13)
- Registering user client connections (line 16)
- Listening to Redis subscribe events (line 19)
- And sending the message to all WebSocket clients (21).

Lines 5 and 6 are simply used to configure the Redis server location and the port to use for the Web Socket server. As you can see it is pretty simple.

Running the WebSocket server

If you have not yet installed `nodemon`, install it now. Then start the WebSocket server using the following command:

```
1  > nodemon server.js
```

004-nodemon-start.sh hosted with ❤ by **GitHub** **view raw**

Let's now create the frontend that will receive the notifications and print them to the user.

Creating the frontend with Vue.js

Open a new terminal and run the following command from the `notifications` directory:

If you have not already installed the Vue CLI tool already, do so now using the command `npm install -g @vue/cli`.

```
1  > vue create webclient
2
3  > cd web-client
```

005-vue-create.sh hosted with ❤ by **GitHub** **view raw**

This command creates a new Vue project that is ready to be executed and extended.

One last package to install for this demonstration is BootstrapVue, which makes it easy to use the CSS library and components from the popular Bootstrap framework.

```
1  > npm install bootstrap-vue bootstrap
```

006-bootstrap-install.sh hosted with ❤ by **GitHub** **view raw**

Open the web-client directory in your favorite code editor, then start the newly created Vue application:

```
1  > npm run serve
```

007-npm-serve.sh hosted with ❤ by **GitHub** **view raw**

The last command starts the Vue development server that will serve the pages and also automatically reloads the pages when you change them.

Open your browser, and go to http://localhost:8080; where you should see the default Vue welcome page:

Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.

Adding WebSocket to the frontend

The Vue framework is quite simple, and for this post we will keep the code as simple as possible. So let's quickly look at the directory structure:

```
├── README.md
├── babel.config.js
├── node_modules
├── package-lock.json
├── package.json
├── public
│   ├── favicon.ico
│   └── index.html
└── src
    ├── App.vue
    ├── assets
    │   └── logo.png
    ├── components
    │   └── HelloWorld.vue
    └── main.js
```

The files at the root level (`babel.config.js`, `package.json`, `package-lock.json`, `node_modules`) are used to configure the project. The most interesting part, at least for now, is located in the `src` directory:

- The `main.js` file is the main JavaScript file of the application, which will load all common elements and call the `App.vue` main screen. We will modify it later to add Bootstrap support.
- The `App.vue` is a file that contains in the HTML, CSS, and JavaScript for a specific page or template. As an entry point for the application, this part is shared by all screens by default, so it is a good place to write the notification-client piece in this file.

The `public/index.html` is the static entry point from where the DOM will be loaded. If you look at it you will see a `<div id="app">`, which is used to load the Vue application.

This demonstration is quite simple, and you will have to modify only two files: the `App.vue` and `main.js` files. In a real-life application, you would probably create a Vue.js component that would be reused in various places.

Updating the App.vue file to show WebSocket messages

Open the `App.vue` file in your editor and add the information listed below. At the bottom of the page, just before the `</div>` tag, add the following HTML block:

```
<hr/>
<h3>
  {{message}}
</h3>
<hr/>
```

009-app-vue.html hosted with ❤ by **GitHub** **view raw**

Using {{message}} notation, you are indicating to Vue to print the content of the message variable, which you will define in the next block.

In the `<script>`, replace the content with:

```
1  <script>
2  import HelloWorld from './components/HelloWorld.vue'
3
4  export default {
5    name: 'App',
6    data() {
7      return {
8        message: "",
9      }
10   },
11   created(){
12     try {
13       const ws = new WebSocket("ws://localhost:3000/");
14       ws.onmessage = ({data}) => {
15         this.message =  data;
16         console.log(this.message);
17       }
18     } catch(err) {
19       console.log(err);
20     }
21   },
22   components: {
23     HelloWorld
24   }
25 }
26 </script>
```

010-app-vue.js hosted with ❤ by **GitHub** **view raw**

These few lines of code are used to:

- Connect to the WebSocket server (line 13)
- Consume messages from the server and send them the local `message` variable (lines 13-17)

If you look carefully at what has been changed, you can see that you have added:

- A `data()` function that indicates to the Vue component that you are defining local variables that can be bound to the screen itself (lines 6-10)
- A `created()` function that is called by the Vue component automatically when it is initialized

Sending messages from Redis to your Vue application

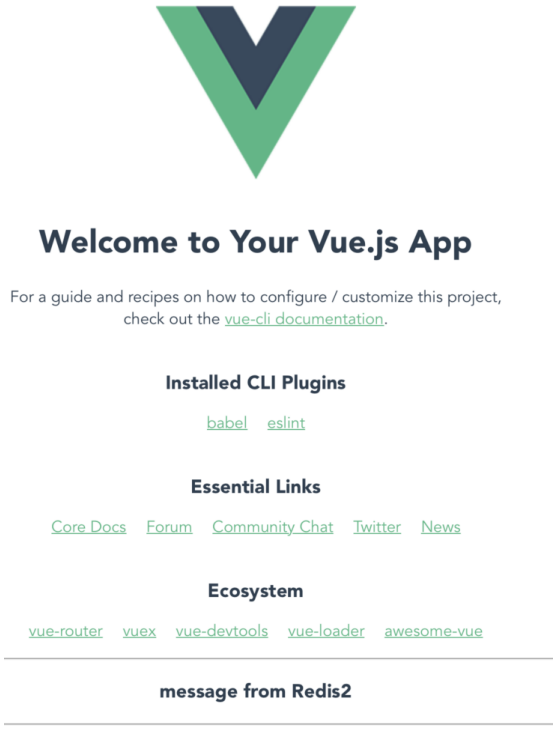
The WebSocket server and the Vue frontend should now be running and connected thanks to the few lines of JavaScript you added. It's time to test it!

Using the Redis CLI or RedisInsight, publish some messages to the `app:notifications` channel. For example, if you started Redis using Docker, you can connect to it using the following command and start publishing messages:

```
1  > docker exec -it redis-server redis-cli
2
3  127.0.0.1:6379> PUBLISH app:notifications  "message from
```

011-docker-redis-cli.sh hosted with ❤ by **GitHub** **view raw**

You should see the message appear at the bottom of the application in your browser:



A Redis message displayed in the view application.

As you can see, it is pretty easy to push content to your web frontend in real time using WebSocket. So now lets improve the design and add a more user-friendly interface using Bootstrap.

Creating an alert block with Bootstrap

In this section, we'll show you how to use a Bootstrap alert component, which appears when a new message is received and disappears automatically after a few seconds, using a simple countdown.

Main.js file

Open the main.js file and add the following lines after the last import:

```
1  import { BootstrapVue } from 'bootstrap-vue';
2  import 'bootstrap/dist/css/bootstrap.css';
3  import 'bootstrap-vue/dist/bootstrap-vue.css';
4
5  Vue.use(BootstrapVue);
```

012-bootstrap.js hosted with ❤ by **GitHub** **view raw**

These four lines import and register the Bootstrap components in your Vue application.

App.js file

In the `App.vue` file, replace the code you added earlier (everything between the two `<hr/>` tags and the tags themselves) with the following:

```
1  <div class="d-inline-flex p-4">
2     <b-alert id="notification"
3        :show="dismissCountDown"
4        dismissible
5        @dismissed="dismissCountDown=0"
```

```
      @dismiss-count-down="countDownChanged"
    >
    New notification: {{message}}
   </b-alert>
</div>
```

013-app.html hosted with ❤ by **GitHub** **view raw**

This component uses several attributes:

- `id="notification"` is the element id used to reference the element in JavaScript or CSS code`:show="dismissCountDown"` indicates that the component is visible only when the `dismissCountDown` variable is not null nor 0
- `dismissible` adds a small icon in the alert to let the user manually close it
- `@dismissed="dismissCountDown=0"` shows that the alert box will be closed then the value `dismissCountDown` equals 0
- `@dismiss-count-down="countDownChanged"` is the countdown method

Let's add a few lines of JavaScript to define the variables and methods used by the alert component:

```

data() {
    return {
      message: "",
      dismissSecs: 5,
      dismissCountDown: 0,
    }
  },
  created(){
    try {
      const ws = new WebSocket("ws://localhost:3000/");
      ws.onmessage = ({data}) => {
        this.message =  data;
        this.showAlert();
      }
    } catch(err) {
      console.log(err);
    }
  },
  methods: {
    countDownChanged(dismissCountDown) {
      this.dismissCountDown = dismissCountDown
    },
    showAlert() {
      this.dismissCountDown = this.dismissSecs
    }
  },

...
```

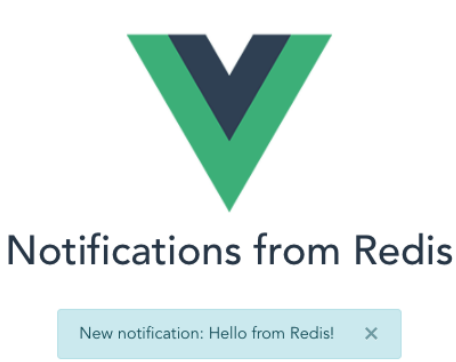
014-data-section.js hosted with ❤ by **GitHub** **view raw**

In this section you have:

- Added the `dismissSecs` and `dismissCountDown` and variables to the data() method (lines 4-5) that are used to control the timer that shows the alert before hiding it again
- Created methods to show and hide the alert component (line 10-26)
- Called the `showAlert()` method when a new message is received (line 13)

Let's try it!

Go back to redis-cli or Redis Insight and post new messages to the `app:notifications` channel.



The notification in an alert box visible in the Vue application.

As you can see, it is easy to use Redis to create a powerful notification service for your application. This sample is pretty basic, using a single channel and server and broadcasting to all the clients.

The goal was really to provide an easy way to start with WebSocket and Redis Pub/Sub to push messages from Redis to a web application. There are many options to deliver messages to specific clients using various channels, and to scale and secure the application.

You can also use the WebSocket server in the other direction, to consume messages as well as to push messages to clients. But that's a big topic for another blog post. In fact, stay tuned for more blog posts on how you can use Redis Gears to easily capture events directly in the Redis database and push some events to various clients.

For more information, see these resources:

- Redis Microservices for Dummies (free ebook)
- Introduction to Redis Streams (Redis documentation)
- Redis Pub/Sub (Redis documentation)
- Take a free online course at Redis University
- Check out Redis Enterprise Cloud