tuts+

CODE > CLOUD SERVICES

Get Started With Pusher: Demonstrating Real-Time Communication With Channels

by Jeremy McPeak 27 Jul 2018

Difficulty: Beginner Length: Short Languages: English

Cloud Services Cross-Platform Web Apps

This post is part of a series called Get Started With Pusher.

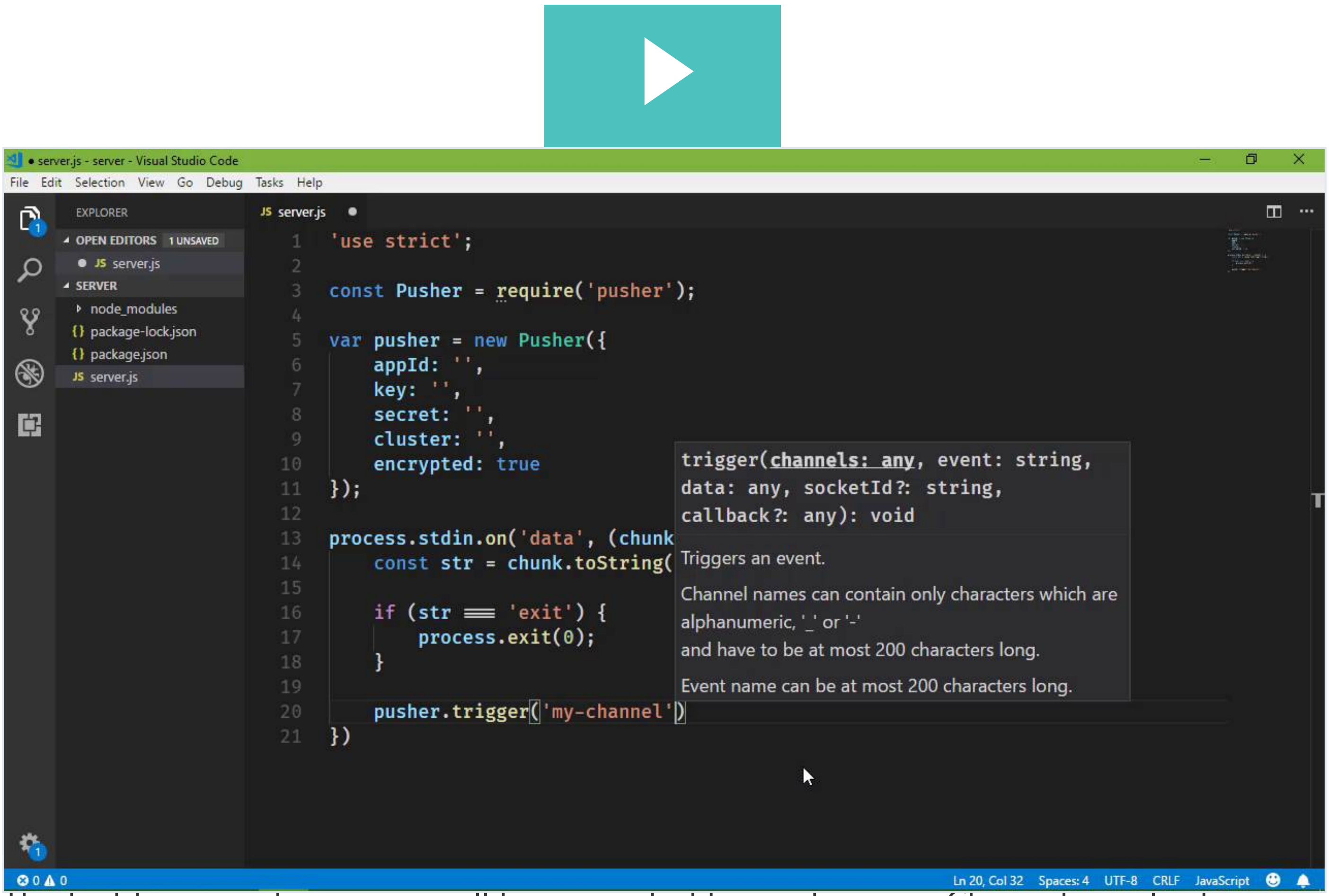
Get Started With Pusher: Introducing Channels

Get Started With Pusher: Build a Chat App With Channels, PHP, and Vue.js

Sponsored Content

This sponsored post features a product relevant to our readers while meeting our editorial guidelines for being objective and educational.

Channels from Pusher is a platform that makes it easy to add seamless real-time data into your apps. In this video, I'll show you how to get started coding real-time communication between client and server apps with JavaScript and Channels from Pusher.


● server.js - server - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS 1 UNSAVED
server.js
SERVER
node_modules
package-lock.json
package.json
server.js
Ln 20, Col 32 Spaces: 4 UTF-8 CRLF JavaScript

```
'use strict';

const Pusher = require('pusher');

var pusher = new Pusher({
    appId: '',
    key: '',
    secret: '',
    cluster: '',
    encrypted: true
});

process.stdin.on('data', (chunk
    const str = chunk.toString(

    if (str === 'exit') {
        process.exit(0);
    }

    pusher.trigger('my-channel')
})
```

```
trigger(channels: any, event: string,
data: any, socketId?: string,
callback?: any): void
```

Triggers an event.

Channel names can contain only characters which are alphanumeric, '_' or '-'
and have to be at most 200 characters long.

Event name can be at most 200 characters long.

The first time you log in, you will be prompted to create a new Channels application. Channels will provide a default name for your application, but it makes more sense to name the Channels application something similar to your application. Because we will write a Node.js console application, I will call my application **node-console-app**. Notice that the naming convention is to use dashes in place of spaces.

Next, you need to choose your cluster, and you want to choose what is closest to your server because clients can be anywhere around the world. For me, it's a toss-up between Ohio and North Virginia. I chose Ohio because that was selected by default.

You then get to choose the technologies you will use to write your app. I chose Node.js

You then get to choose the technologies you will use to write your app. I chose Node.js, but feel free to use whatever technologies you want.

After clicking the **Create my app** button, you'll see the **Getting Started** page. This page is a client, and notice it says the connection state is connected. If we look at the **Overview** page, we see that there is one client. That is this demo client on the Getting Started page.

At the bottom of the Overview page, you'll find the app id, the key, the secret key, and the cluster. This is the information you need to connect to your Channels application from within your client and server applications.

Creating the Server App

In a new directory, create a **package.json** file with the following command:

```
npm init --yes
```

You'll then want to install the Pusher package.

```
npm install pusher --save
```

Next, create a file called **server.js**—this is our application file. Enter the following code:

```
'use strict';



const Pusher = require('pusher');



var pusher = new Pusher({

    appId: '530620',

    key: 'b534d4fac76717b9872e',

    secret: 'f84f62e45f82cc09b8c8',

    cluster: 'us2',

    encrypted: true

});
```

This code creates a `Pusher` object by passing an object that has properties for the app id, the key, the secret key, and the cluster to the constructor—basically all of the information that we just saw on the Overview page is going to be here. There's also another property called `encrypted` which specifies that Pusher should encrypt the communication between our server app and the Channels service.

Our simple server application will accept user input by allowing us to type into the console window. We'll grab that input and then trigger a messaging event. That code looks like this:

```
process.stdin.on('data', (chunk) => {

    const str = chunk.toString().trim();



    if (str === 'exit') {

        process.exit(0);

    }



    pusher.trigger('my-channel', 'my-event', {

        message: str

    });

});



console.log('Type a message...');
```

In this code, we use the standard input stream and listen for the data event. The data we receive is raw data, so we then convert it to a string and trim the whitespace. We then check to see if the user typed the word "exit" and, if so, we exit the program.

We then use the `Pusher` object's `trigger()` method to trigger the `my-event` event in the `my-channel` channel. So triggering an event involves the following three pieces of information in order:

- the channel
- the event
- the message payload

It’s important to note the channel name is not the name of the Channels app. Rather it’s an arbitrary name that hopefully has some kind of significance to our application. We used the default, `my-channel`, because that is what the demonstration client on the Getting Started page is subscribed to. Clients subscribed to this channel can then listen for events that occur in the `my-channel` channel. In this case, they’ll need to listen for the `my-event` event, because that's what we are triggering when we enter something in the Node.js application’s console.

The message payload can be an object of any shape. Once again, this code sets a message property because that is what the Getting Started client is looking for.

We finished off our app code by outputting a message that tells the user that the application is ready.

Advertisement

Testing the Server Code

You can immediately test your server app because we already have a client: the Getting Started page. You can run the server with the following command:

```
node server.js
```

Messages you type in the console app should display in an alert box on the Getting Started page. Feel free to play around with that before we move on to writing the client app in the next section.

Writing the Client App

Initialize the client project in another directory with the following command:

```
npm init --yes
```

Then install the client Pusher library with the following command:

```
npm install pusher-js --save
```

Create a file called **client.js** and type the following code:

```
'use strict';

const Pusher = require('pusher-js');

let pusher = new Pusher('b534d4fac76717b9872e', {
    cluster: 'us2',
    encrypted: true
});
```

```
let channel = pusher.subscribe('my-channel');

channel.bind('my-event', (data) => {
    console.log(data.message);
});

console.log('Listening for messages...');
```

This code creates a client `Pusher` object. Notice that the client object requires less information than the server—*don't include the secret key or the app ID in your client code!* In our example, we then subscribe to the `my-channel` channel using the `Pusher` object's `subscribe()` method. This allows you to listen for any event in that channel.

Subscribing to a channel gives you a channel object that you can then use to listen for events that occur in that channel, and in this code, we used the `bind()` method to bind a listener to the `my-event` event. Every time the client handles the `my-event` event, it uses `console.log()` to write the message to the screen.

Go ahead and run the client in a separate console window so that you can run both the client and server at the same time. The command to run the app is:

```
node client.js
```

Type messages into the server application, and you should see the client receive and output them.

Conclusion

Channels is a fantastic platform that gives you the ability to add real-time communication to your apps, and as you saw from the two applications we wrote in this video, you can add the power of real-time communication with relative ease using Channels and their libraries.

Jeremy McPeak

I started my development career on the client-side writing JavaScript and DHTML components in my spare time. In 2005, Nicholas C. Zakas asked me to join him in writing the first edition of Professional Ajax for Wiley Publishing. Since Professional Ajax, 1st Edition, I've been blessed to take part in other book projects: Professional Ajax 2nd Edition, and Beginning JavaScript 3rd and 4th editions.