Stream Audio into a Phone Call with Node.js

Last updated on May 12, 2021

When you have your customer on a voice call, you have her undivided attention. Why not use that opportunity to tell her the latest news about your company, relay an inspiring message from your CEO, or even just play her your latest advertising jingle?

In this blog post, you will learn how to play an audio file into an active call programmatically, using the Vonage Voice API and Node.js.

Before you Begin

To work through this example you'll need Node.js. If you don't already have it installed, download it from the Node.js website.

Vonage API Account

To complete this tutorial, you will need a Vonage API account. If you don't have one already, you can sign up today and start building with free credit. Once you have an account, you can find your API Key and API Secret at the top of the Vonage API Dashboard.

This tutorial also uses a virtual phone number. To purchase one, go to *Numbers* > *Buy Numbers* and search for one that meets your needs.

You can provision a number in the Developer Dashboard, but we'll talk you through using the Vonage CLI to rent a number, create a voice application and then link your number to it.

Finally, you'll want the source code, which is available on GitHub. Clone the repository and `cd` into the application's root directory.

Installing the Vonage CLI

Install the Vonage CLI globally using the following command:

```
npm install @vonage/cli -g
```

Then, configure the CLI with your Vonage API key and secret, which you will find in the Developer Dashboard:

```
vonage config:set --apiKey=VONAGE_API_KEY --apiSecret=VONAG
```

Replace the `VONAGE_API_KEY` and `VONAGE_API_SECRET` with your own details to authenticate the CLI.

Renting a Vonage Number

You need a number to make calls from. Rent one by executing the following command, replacing the country code as appropriate. For example, if you are in the USA, replace `GB` with `US`:

```
vonage numbers:search US
vonage numbers:buy [NUMBER] [COUNTRYCODE]
```

Make a note of the telephone number that the command returns.

Creating the Voice Application

To use the Voice API, you must create a Voice API application. This is not the same thing as the web application you are building. It is merely a container for the configuration and security information you need to connect to Vonage's APIs.

Create an application using the CLI and make a note of the application ID it returns:

```
vonage apps:create "Play audio app"  --voice_answer_url=ht
```

Note that the `apps:create` command shown uses a few parameters. These are to set the webhook endpoints and generate your private key to authenticate your application.

The Vonage APIs need to know your webhook endpoints so that they can make requests to them when there is an incoming call or an event that your application should know about. You can safely leave these as `example.com` because we will specify the webhooks programmatically.

Initialize the Dependencies

The application has the following external dependencies:

- **`dotenv`** - a module that allows us to easily configure the application using a **`.env`** file
- **`express`** - a lightweight web framework
- **`body-parser`** - middleware to handle **`POST`** requests
- **`vonage`** - the REST client library for Node.js

These dependencies are configured in `package.json`. Run `npm install` to install them to the `node_modules` subdirectory.

Make your Application Available on the Public Internet

We need to expose our application to the Internet so that Vonage's servers can access our webhooks. We recommend using ngrok for this.

Follow the instructions in this blog post to install and run ngrok on port 3000. For example:

ngrok will give you a temporary URL such as `https://914288e7.ngrok.io`. Make a note of it.

Leave ngrok running while you are using the application, otherwise, the URLs will change and you will need to reconfigure it.

Configure the Environment

Copy `example.env` to `.env` and enter the details you have harvested from the above steps:

```
VONAGE_API_KEY= Your Vonage API key
VONAGE_API_SECRET= Your Vonage API secret
VONAGE_APPLICATION_ID= The Voice API application ID
VONAGE_APPLICATION_PRIVATE_KEY_PATH=private.key
VONAGE_NUMBER= The number you rented from Vonage
TO_NUMBER= The number you want the application to call
BASE_URL= The Ngrok temporary URL
```

Then, change the path to the audio file in the `answer.json` package to match your ngrok hostname:

```
...
    {
        "action": "stream",
        "streamUrl": [
            "http://914288e7.ngrok.io/audio/silence.mp3"
        ]
    }
]
```

You are now ready to see the application in action!

Run the Application

Launch the application by executing the following command:

Visit `http://localhost:3000/call` in your browser. This makes a `GET` request to the `/call` endpoint in your application and causes it to ring the `TO_NUMBER` in `.env`.

You will hear a message, followed by some music that plays for 20 seconds, and then the call disconnects.

How It Works

All the logic happens in `server.js`.

Instantiating the Vonage Client

The application first reads the settings you configured in

`.env` into environment variables and uses them to instantiate the Vonage REST API client library for Node.js:

```
const TO_NUMBER = process.env.TO_NUMBER
const VONAGE_NUMBER = process.env.VONAGE_NUMBER
const BASE_URL = process.env.BASE_URL

const VONAGE_API_KEY = process.env.VONAGE_API_KEY
const VONAGE_API_SECRET = process.env.VONAGE_API_SECRET
const VONAGE_APPLICATION_ID = process.env.VONAGE_APPLICATIO
const VONAGE_APPLICATION_PRIVATE_KEY_PATH = process.env.VON

const vonage = new Vonage({
  apiKey: VONAGE_API_KEY,
  apiSecret: VONAGE_API_SECRET,
  applicationId: VONAGE_APPLICATION_ID,
  privateKey: VONAGE_APPLICATION_PRIVATE_KEY_PATH
})
```

Serving the Audio Files and NCCO

We use two audio files in this application. One is for the music that we play to the caller. The other is a silent MP3 file. You can only play audio into an existing call, so it's important that we keep the call open otherwise it will disconnect before you have a chance to do anything with it. The silent MP3 file keeps the call open.

The welcome message and silent MP3 are defined in a Call Control Object (NCCO). The NCCO is an array of

JSON objects that represents the actions that occur within the call. Our `answer.json` NCCO tells the application to read some text and then stream a silent MP3:

```
[
    {
        "action": "talk",
        "text": "<speak>Please hold to listen to some music
        "voiceName": "Russell"
    },
    {
        "action": "stream",
        "streamUrl": [
            "http://914288e7.ngrok.io/audio/silence.mp3"
        ]
    }
]
```

This is one way of streaming audio into a call. But the focus of this post is how to do it *programmatically*.

These resources are served from the application's `public` directory as follows:

```
app.use('/audio', express.static(path.join(__dirname, 'publ

const answer_url = BASE_URL + '/audio/answer.json'
const audio_url = BASE_URL + '/audio/music.mp3'
const event_url = BASE_URL + `/webhooks/events`
```

Making the Outbound Call

When we make a `GET` request to the `/call` route, the application uses the `vonage` client library's `calls.create()` method to make an outbound call to the `TO_NUMBER` configured in `.env`. When the call is answered, it responds with the actions defined in the `answer.json` NCCO and starts listening to Voice API events on the `/webhooks/events` webhook handler:

```
app.get('/call', makeOutboundCall)

const makeOutboundCall = (req, res) => {
  console.log('Making the outbound call...')

  vonage.calls.create({
    to: [{
      type: 'phone',
      number: TO_NUMBER
    }],
    from: {
      type: 'phone',
      number: VONAGE_NUMBER
    },
    answer_url: [answer_url],
    event_url: event_url
  })
}
```

Playing Audio into the Call

In the event webhook, we check the call `status`. If it is `answered`, then we retrieve the call ID so that we can play the audio into the correct call and stop playing the audio after 20 seconds. After the audio is stopped, we use the `vonage.calls.update()` method to replace the existing NCCO `action` with a new one: `hangup`, which disconnects the call:

```
app.post('/webhooks/events', (req, res) => {
  if (req.body.status == 'answered') {

    const call_uuid = req.body.uuid



    start_stream(call_uuid)



    setTimeout(() => {
      stop_stream(call_uuid)
      vonage.calls.update(call_uuid, { action: 'hangup' },
        console.log("Disconnecting...")
      });
    }, 20000)



  }
  res.status(200).end()
})
```

The `start_stream()` function uses the Vonage client library's `calls.stream.start()` method to programmatically play the specified audio file into the call:

```
const start_stream = (call_uuid) => {
  console.log(`streaming ${audio_url} into ${call_uuid}`)
  vonage.calls.stream.start(call_uuid, { stream_url: [audio
    if (err) {
      console.error(err)
    }
    else {
      console.log(res)
    }
  })
}
```

The `stop_stream()` function calls `vonage.calls.stream.stop()` to stop playing the audio file:

```
const stop_stream = (call_uuid) => {
  vonage.calls.stream.stop(call_uuid, (err, res) => {
    if (err) {
      console.error(err)
    }
    else {
      console.log(res)
    }
  })
}
```

Conclusion

In this post, you learned how to play audio into an existing call and about some of the NCCO `actions` that govern the call flow. Feel free to experiment by substituting different audio files and NCCO `actions`. The following resources might help:

- Voice API reference
- NCCO reference
- Using SSML in TTS (text-to-speech)
- The REST API client library for Node.js