Last week I had the distinct pleasure of being on my buddy Todd Sharp's live stream, Streaming on Streaming. You can watch the recording of that session here:

Todd is the principal developer advocate for Amazon Interactive Video Service (IVS), or, more simply, a way of taking the *incredible* power of the Twitch platform and putting it in your hands. IVS gives you:

- The ability to create channels for your content
- Multiple ways to broadcast your content, from web-based to more powerful tools like OBS.
- Multiple ways to display your content, including a pretty simple web SDK
- Super detailed reporting
- And deep integrations into the stream that allows for things like transcriptions, Lambda-based chat moderation, and more.

This is a commercial service but like pretty much everything in AWS, there's a free tier that lets you test things out and see if it makes sense for you.

In the session I had with Todd, he wanted to put their own documentation and console through a "first-time developer's experience." So first off, thinking about and really caring about the initial developer experience with your product is probably one of the most important things you can do. I know in my own career in developer advocacy, it's near impossible for me to try something new without constantly examining and commenting on the developer onboarding. Todd deserves *mad* respect for doing this live on his stream. I'm kinda well known for breaking things, doing things wrong, and generally just being your worst nightmare when it comes to DX, so that was quite brave on his part. (And to be clear, we didn't preplan any of this stream. I wasn't given anything in advance, and the only real knowledge I had was reading his blog posts and general chat.)

As you can imagine, something like "roll your own IVS" is going to be really complex, but in our one-hour session, we got my channel setup, I broadcasted from OBS and a web tool and was able to build a simple web page to display the stream. I thought I'd share some of the highlights from that here as the idea of supporting streaming in the Jamstack sounds incredibly compelling. This will be somewhat high-level, but

the docs are pretty thorough and will give you more detail about the particulars. You will also want to start off with the Getting Started guide.

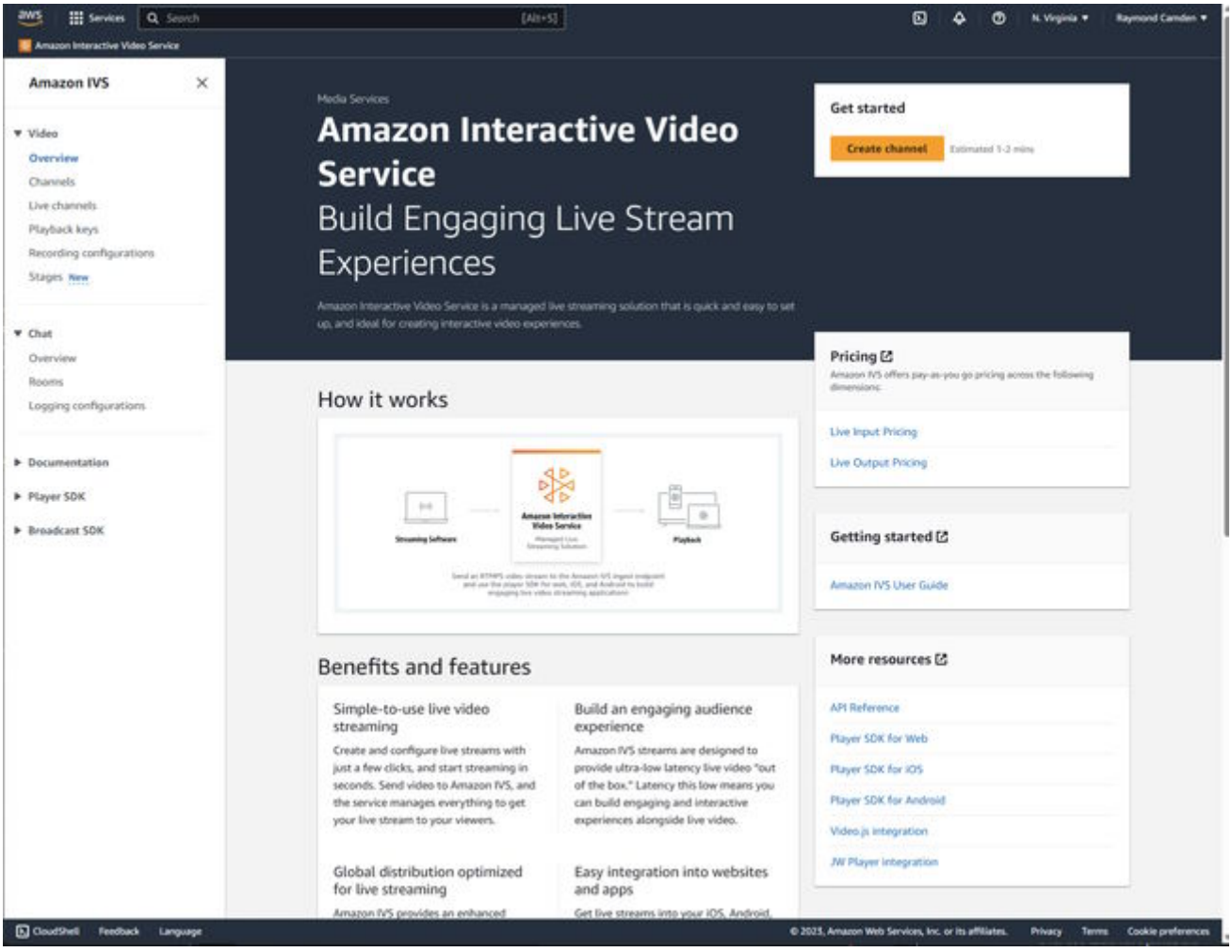
With that in mind, here's a basic outline of how simple this is to get started.

Step One - Get AWS

I'm just going to skip this. I mean I guess I'm not as I'm calling it out, and you should never assume anything, but as a large percentage of folks already use AWS, my assumption is that you already have an account. I did and used my root one, but if you follow the getting started link I shared above, they have you set up a user with more restricted access, which is The Right Thing to Do, and We Always Do The Right Thing in tech. Ahem.

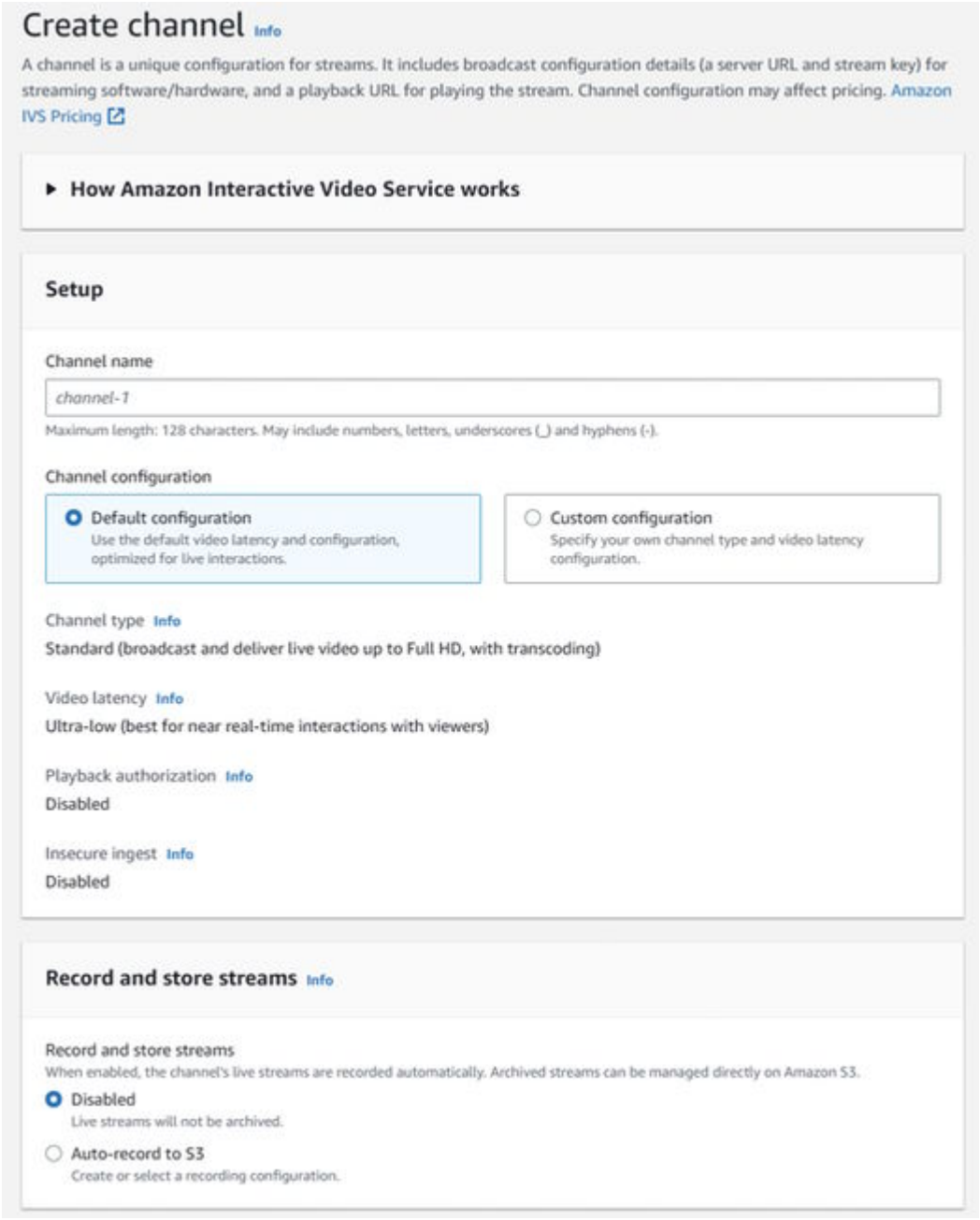
Step Two - Make Your Channel

The next thing you'll do is define a channel in the IVS portion of the AWS console:



As a reminder, the search box on tox of IVS does a *really* good job of finding stuff. AWS can be a bit overwhelming, but their search has made things a lot easier lately.

I kind of assume most folks know about streaming and get the basic idea of a channel, but if I were to start streaming, I'd probably only have one channel for myself. If I wanted to add streaming to my company's website, I could imagine having a channel for things like training, external events, and so forth. At the bare minimum, one channel is needed. When creating a new channel, the bare minimum is the name:

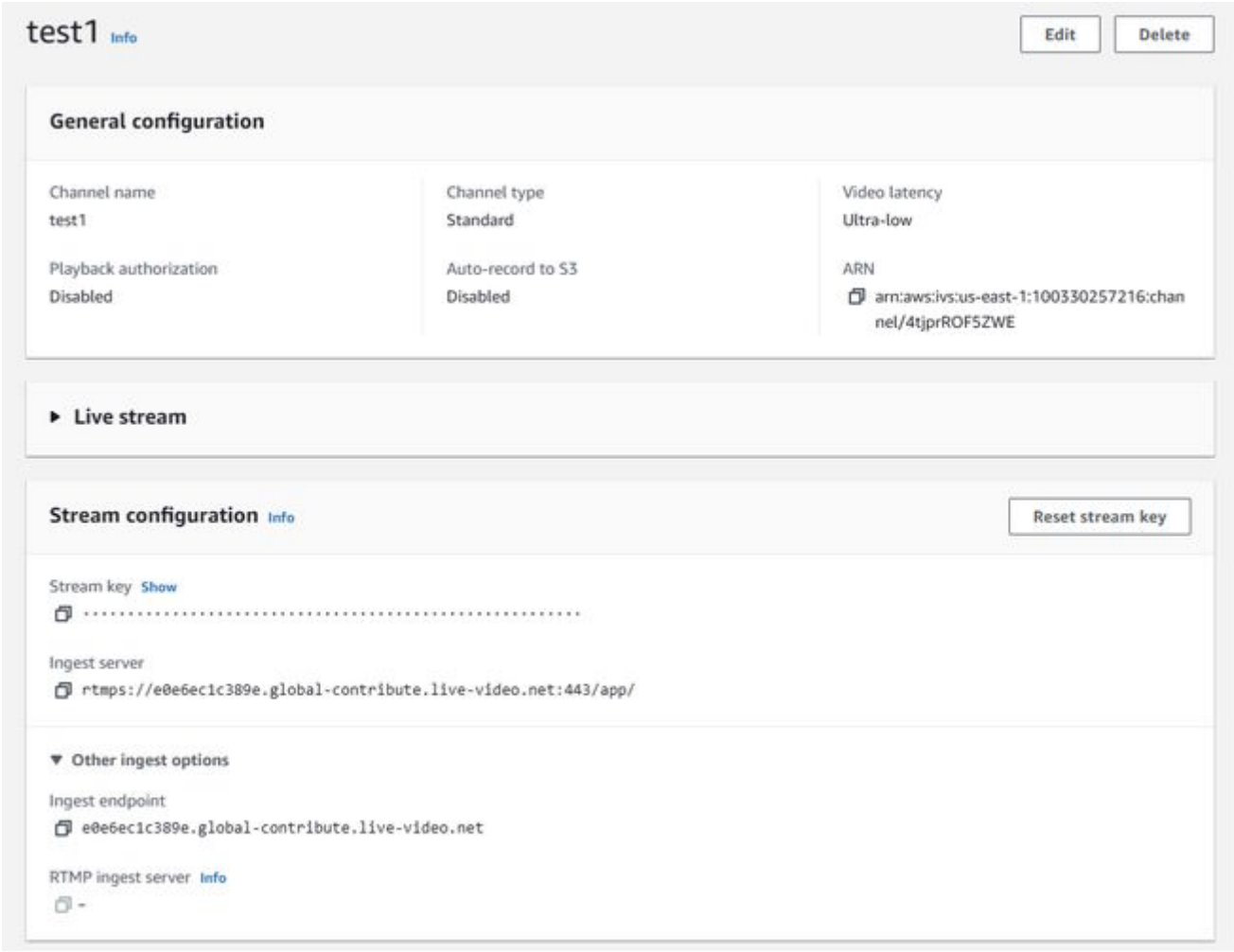


Note that one of the options is automatically storing the streams to S3. I didn't enable that on the stream with Todd, but it's really nice that it's that simple to enable. Obviously, there's going to be a cost in storing large video files, but I appreciate how simple it is to enable.

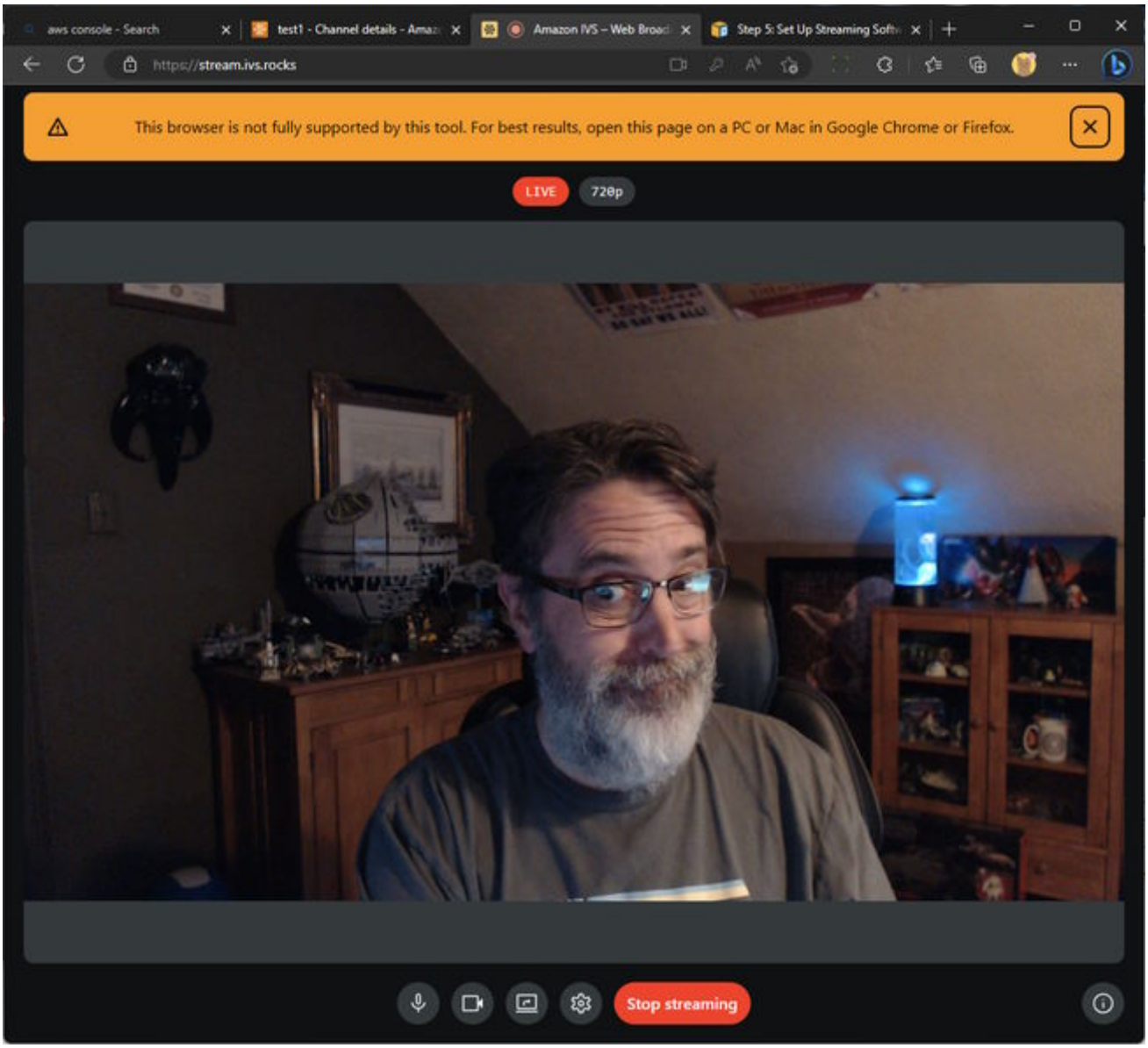
Step Three - Figure out your Broadcaster

The getting started docs concerning setting up your streaming cover using their SDKs, OBS Studio, or FFmpeg. I've used OBS Studio in the past, so during the stream with Todd, I used that. I'll warn folks though that if you've never used it before, it can be quite overwhelming. It definitely was for me (heck, I still barely know what I'm doing). Instead, let me share another option Todd shared with me later in the stream, stream.ivs.rocks. This is a web-based streaming solution that you can use to test IVS.

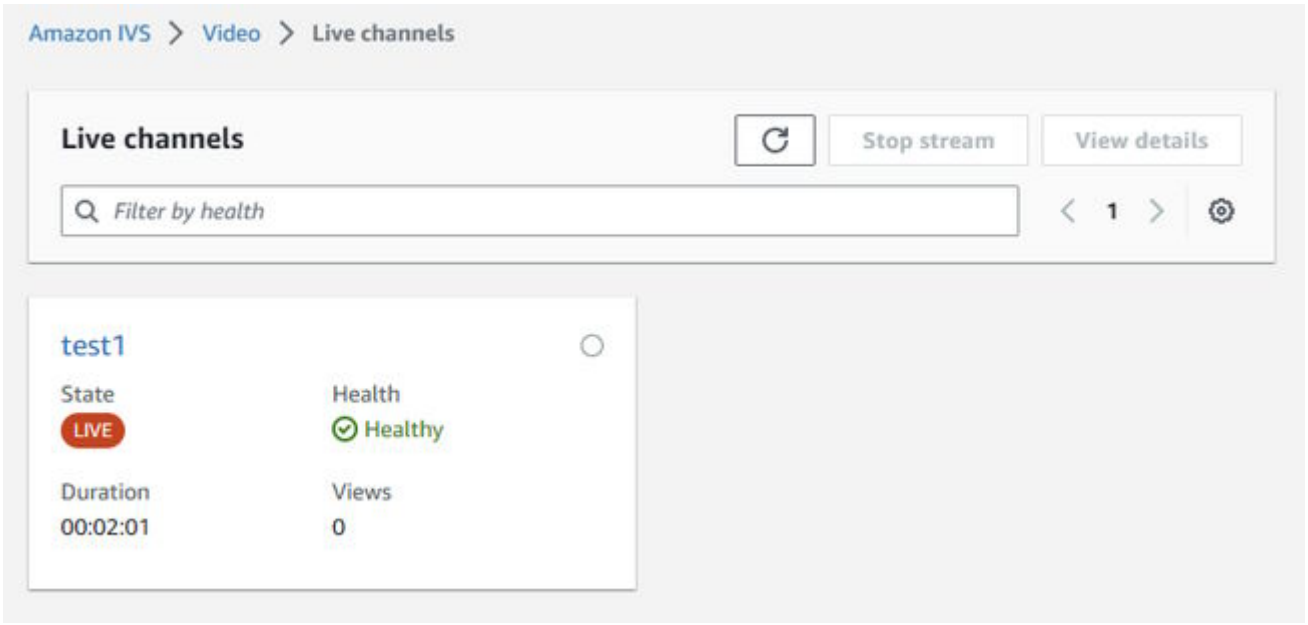
Open the page in your browser (note that it says Edge is not supported, but I had no issues with it), and click the gear icon to get to your settings. You'll want to specify your "Ingest endpoint" and "Stream key", both of which you can find in the AWS console in your channel details. Don't make the same mistake I did - the "Ingest server" is NOT the same as the "Ingest endpoint", you'll want to open and expand "Other ingest options" to see that.



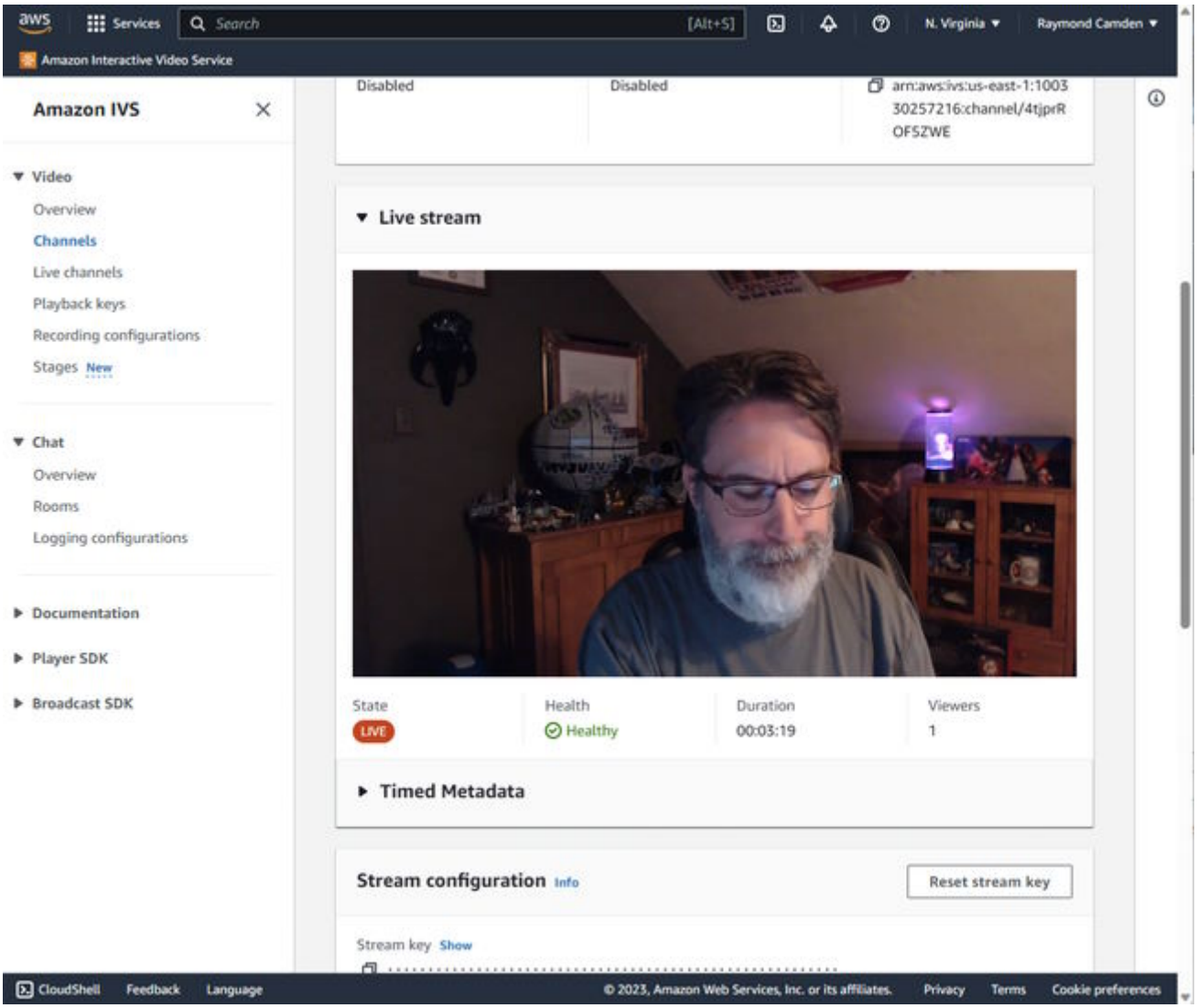
Once you've done that, you can click the "Start streaming" button below, and literally, that's it! While this is not something you would do for a "real" stream, it's *much* simpler and quicker to set up.



Once you're streaming, go back to the AWS console, click on "Live channels", and you can see your stream in action:



If you go to the channel details, it's also displayed there as well:



By the way, while there, and in the future, you can view past streams and all kinds of cool metrics about the stream. A lot of this used terminology I'm not familiar with, so that's something to keep in mind if your new to streaming (and is something I shared with Todd as feedback), but if you like stats, AWS has got you covered.

Step Four - The Front End

Actually viewing the stream requires figuring out *where* you're going to do so. Considering I'm a web guy, and this post is about doing it on the Jamstack, the web guide on their documentation is where you would want to go next. Once again, the initial implementation can be very simple. Here's an entire implementation:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title></title>
    <script src="https://player.live-video.net/1.18.0/amazon-ivs-
player.min.js"></script>
    <style>
    video#video-player {
        width: 800px;
```

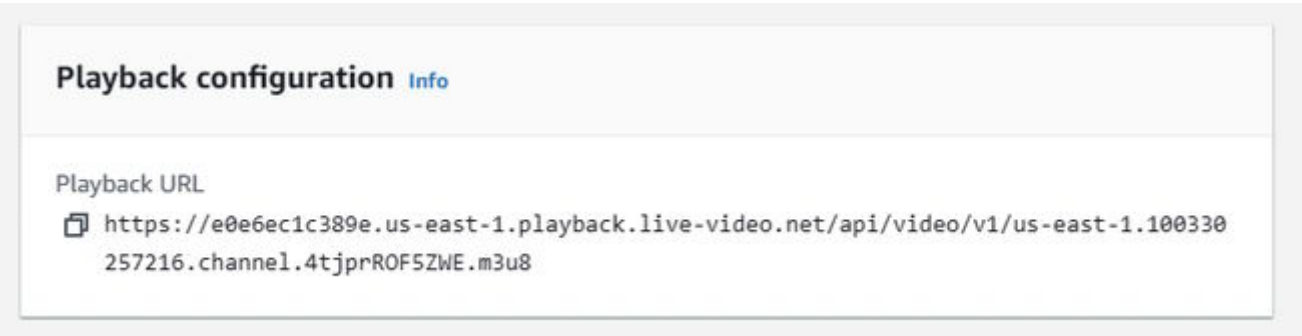
```
            height: 450px;
        }
        </style>
    </head>
    <body>

    <h2>My Stream</h2>
    <video id="video-player" playsinline controls autoplay></video>

    <script>
    const pbURL = 'https://e0e6ec1c389e.us-east-1.playback.live-
    video.net/api/video/v1/us-east-1.100330257216.channel.4tjprROF5ZWE.m3u8';
    document.addEventListener('DOMContentLoaded', init, false);
    async function init() {
        if (IVSPlayer.isPlayerSupported) {
            const player = IVSPlayer.create();
            player.attachHTMLVideoElement(document.getElementById('video-
    player'));
            console.log('player made');
            player.load(pbURL);
            player.play();
        } else console.log('u stink');
    }
    </script>
    </body>
    </html>
```

Let's break it down. You've got a script tag up top loading their SDK. Next, I've got my HTML with the important bits being the `<video>` tag and my CSS to properly size it.

The JavaScript creates an instance of the `IVSPlayer` and attaches it to the DOM. The `pbURL` value comes from the AVS channel config:



And here it is in action!

Now, before getting too excited, note that this is the simplest code you can use, but is not very bulletproof, in fact, and you'll definitely run into this while testing, if you aren't streaming, you get an error trying to load the stream. That *absolutely* makes sense, so better code would handle that. I took a quick look at the web reference and I see event support so you should be able to have it listen for a stream beginning and either load at that point or do something in the DOM to let the person viewing the web page know that a stream began.

That's It!

Ok, sure, that's not very production-worthy, but I literally had a streaming solution done in an hour. Now, this isn't going to be free, so that's something to keep in mind, but I'm really blown away by how quickly you can get started, as well as how much power you get out of the box. And with AWS handling everything, it's a great fit for the Jamstack. There are server-based APIs you could integrate as well, and that would be where I'd use something like Netlify Functions.

So again - check out the docs. There's going to be a lot to digest, but there's a lot more there than I've touched on here.

Next, check out Todd's posts on dev.to, where he's shared a bunch of examples. His posts involving chat and the power you have to moderate there are my favorite I think. I'll also recommend the one he wrote on creating your own "LoFi" radio station as being especially cool.