Building a Livestreamed Murder Mystery Game with Amazon IVS and Timed Metadata

By: Bernie Marger

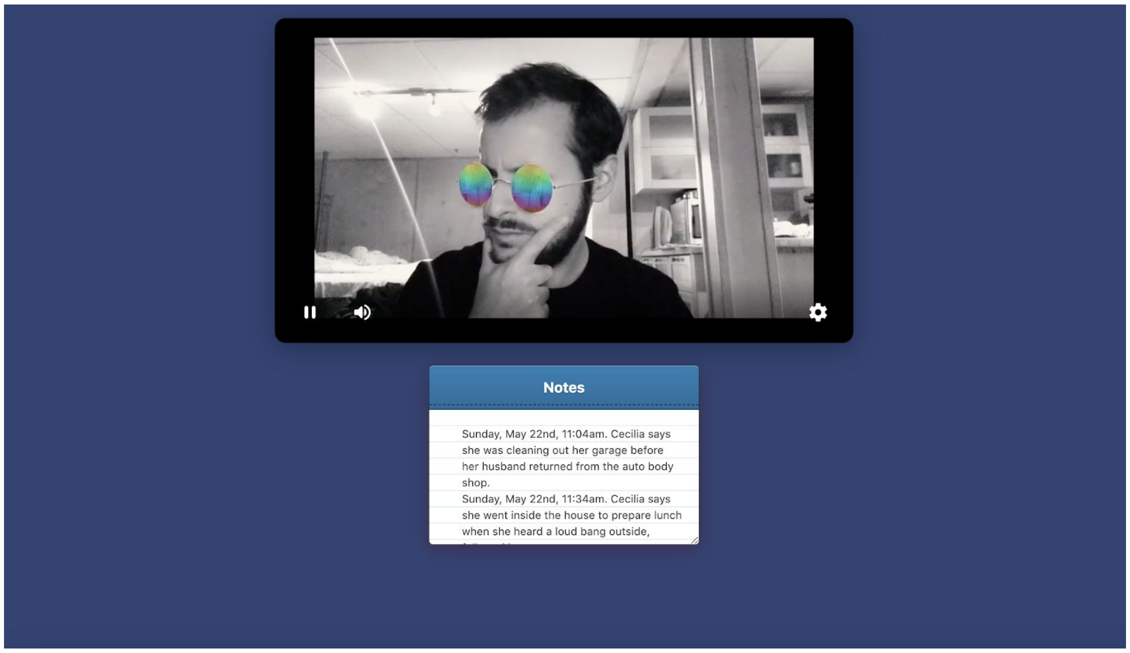
Requirements
Postman
Broadcasting Software
Amazon Web Services (AWS) Account with Billing

Introduction

Happy October 🎃! In this tutorial, we are going to create a livestreamed murder mystery game using Amazon's Interactive Video Service (IVS) and its Timed Metadata API.

For this project, the livestreamed content will be a collection of fictional character monologues framed as police interviews, filmed after a heinous murder was discovered to have taken place. As each character tells their side of the story, clues and hints will begin filling out the viewer's notepad. At the end of the stream, viewers will be able to flip through their copy of the notepad and try to solve the mystery!

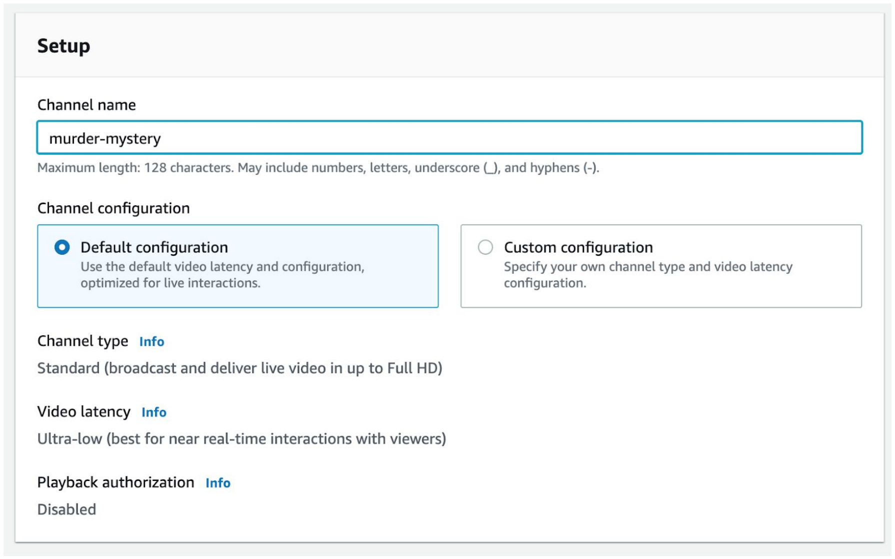
You can find the complete sample code for this project here.



Sunday, May 22nd, 11:04am. Cecilia says she was cleaning out her garage before her husband returned from the auto body shop.
Sunday, May 22nd, 11:34am. Cecilia says she went inside the house to prepare lunch when she heard a loud bang outside,

Creating a Channel in Amazon IVS

We'll begin by setting up our livestream in Amazon IVS. The first step is to log in to the AWS Console and navigate to the Amazon IVS service. Once we're there, let's create a new Channel with a name like *murder-mystery* that retains all of its default settings.



Setup

Channel name

murder-mystery

Maximum length: 128 characters. May include numbers, letters, underscore (_), and hyphens (-).

Channel configuration

[✓ Default configuration] Use the default video latency and configuration, optimized for live interactions.

Custom configuration — Specify your own channel type and video latency configuration.

Channel type Info

Standard (broadcast and deliver live video in up to Full HD)

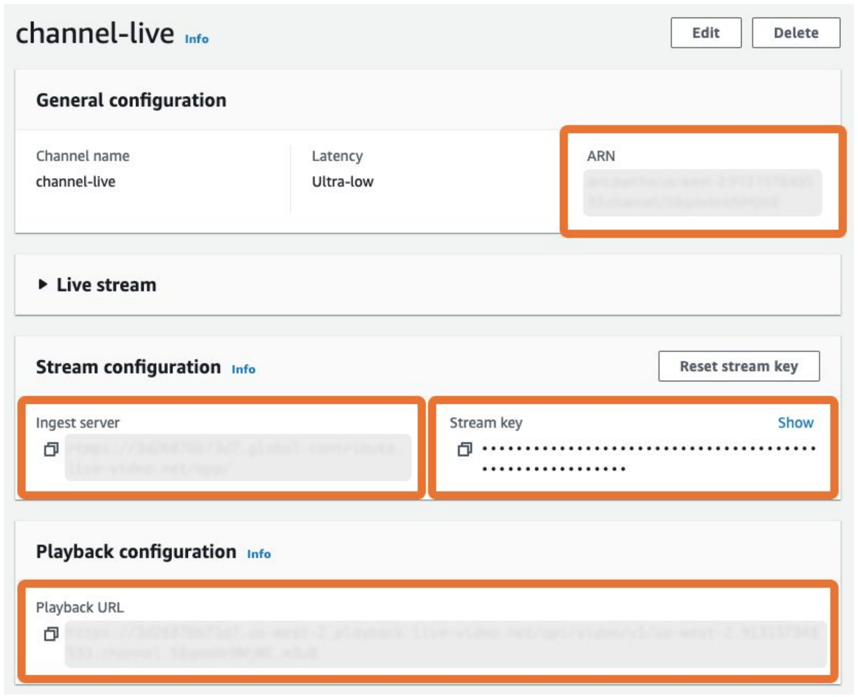
Video latency Info

Ultra-low (best for near real-time interactions with viewers)

Playback authorization Info

Disabled

Once that's created, take note of your Channel's ARN, Ingest Server, Stream Key, and Playback URL.



channel-live Info

General configuration

Channel name	Latency	ARN
channel-live	Ultra-low	[redacted]

▸ Live stream

Stream configuration Info

Ingest server: [redacted]

Stream key: •• Show

Playback configuration Info

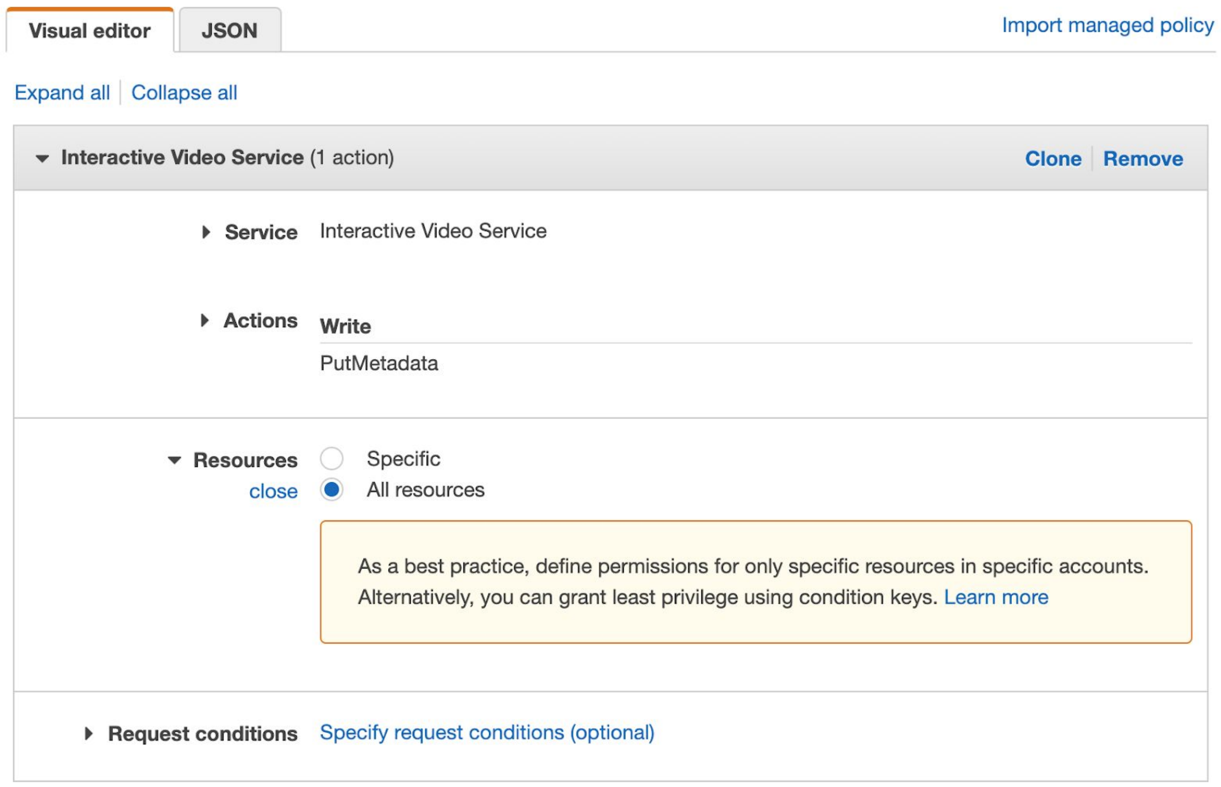
Playback URL: [redacted]

At this point, go ahead and enter your Stream Key and Ingest Server address into your preferred broadcasting software and try to stream to your channel. The video feed should be viewable directly from the “Live stream” section of your Channel’s detail view on the IVS Console.

Gaining AWS Authentication Credentials with IAM

For this project, we will also need a way to make API requests to AWS. In order to do this, we need a set of credentials that are authorized to make requests against specific AWS endpoints - in this case, IVS’s *PutMetadata* endpoint. Luckily, AWS has a way to create this set of credentials via its Identity and Access Management (IAM) service.

First, navigate to IAM via the AWS console and create a new Policy. You’ll need to attach a permission for the Interactive Video Service that allows you to use the PutMetadata action against all resources. In the future, we may want to make this policy more restrictive such that we only allow actions to be taken against specific resources.



Interactive Video Service (1 action)

Service Interactive Video Service

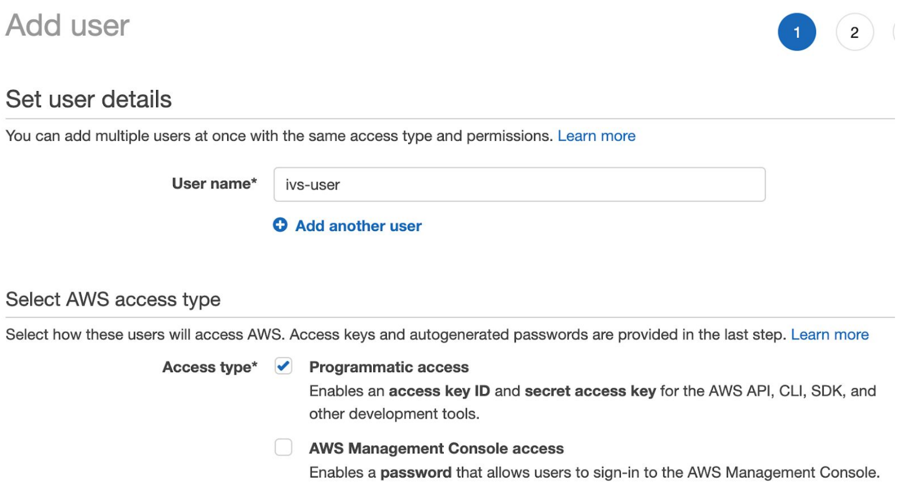
Actions **Write** PutMetadata

Resources ( ) Specific [✓ All resources]

As a best practice, define permissions for only specific resources in specific accounts. Alternatively, you can grant least privilege using condition keys. Learn more

Request conditions Specify request conditions (optional)

Save the Policy under a name you’ll remember, like *murder-mystery-policy.* Next, you’ll create an IAM User to which we’ll apply this Policy. When you create your IAM User, you only need to grant it the Programmatic Access Type as we’ll only be needing it for AWS API authorization.



Add user

1 2

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* ivs-user

Add another user

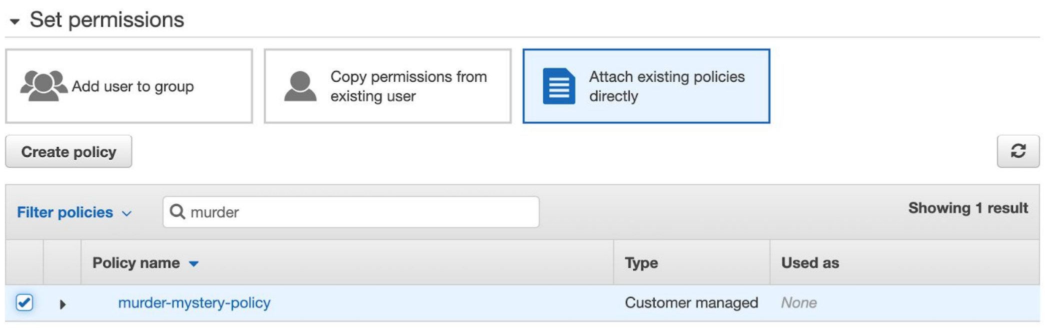
Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

Then attach the policy you just created.



Set permissions

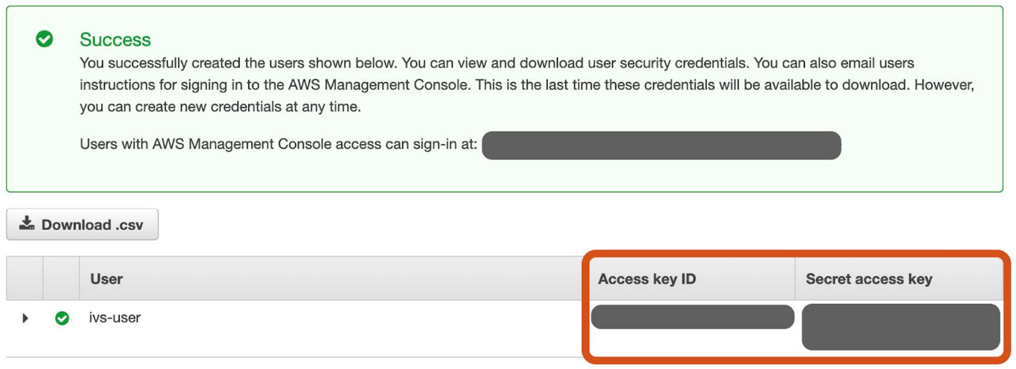
Add user to group | Copy permissions from existing user | [✓ Attach existing policies directly]

Create policy

Filter policies: murder — Showing 1 result

		Policy name	Type	Used as
✓		murder-mystery-policy	Customer managed	None

And on the confirmation screen, you will have a chance to record the Access Key ID and Secret Access Key for your new IAM User. Do so, as we will need this information later.



Success
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: [redacted]

Download .csv

		User	Access key ID	Secret access key
	✓	ivs-user	[redacted]	[redacted]

Building our Murder Mystery Web App

Now that our AWS setup is done, take a moment to clone the Amazon IVS Basic Web Sample Github repository. We will be using the custom-player-controls demo as a boilerplate for our new murder mystery web app. Change the `playbackUrl` in *custom-player-controls/script.js* to be the Playback URL for your new IVS Channel. Now begin streaming to your channel and open *custom-player-controls/index.html* in your browser: you should see your livestream displayed on the custom video player!

Now that we have our livestream embedded in our page, we want to add a representation of the viewer's notepad underneath it. This notepad will fill out with hints and clues as the stream progresses, thanks to IVS's Timed Metadata API. For our notepad, we are going to be using this great Codepen as a solution (credits to Dan Eden, uploaded by Reza Qorbani). Copy all the CSS that affects the *notepad* class and add it to *custom-player-controls/style.css*. Now let's add the notepad in a new div underneath our player.

```
<div class="player-wrapper">
    ...
</div>
<div class="notepad">
    <h1>Notes</h1>
    <textarea id="notepad" disabled></textarea>
</div>
```

Now let's add a container div around both the *player-wrapper* and the *notepad*.

```
<div class="container">
    <div class="player-wrapper">
        ...
    </div>
    <div class="notepad">
        <h1>Notes</h1>
        <textarea id="notepad" disabled></textarea>
    </div>
</div>
```

Set that *container* with a flexbox style.

```
.container {
    height: 100%;
    width: 100%;
    padding-top: 1rem;
    display: flex;
    flex-direction: column;
    align-items: center;
}
```

And ensure that the *player-wrapper* never grows beyond a fixed 16x9 aspect ratio.

```
.player-wrapper {
    display: flex;
    flex-grow: 1;
    width: 100%;
    max-width: 640px;
    max-height: 360px;
    ...
}
```

Sending Clues with Timed Metadata and Postman

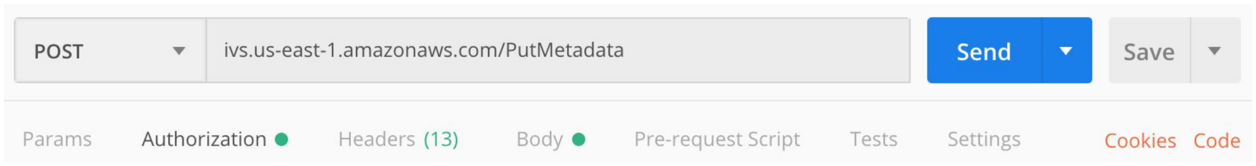
Now it's time to set the contents of our *notepad* automatically using IVS Timed Metadata.

What is Timed Metadata?

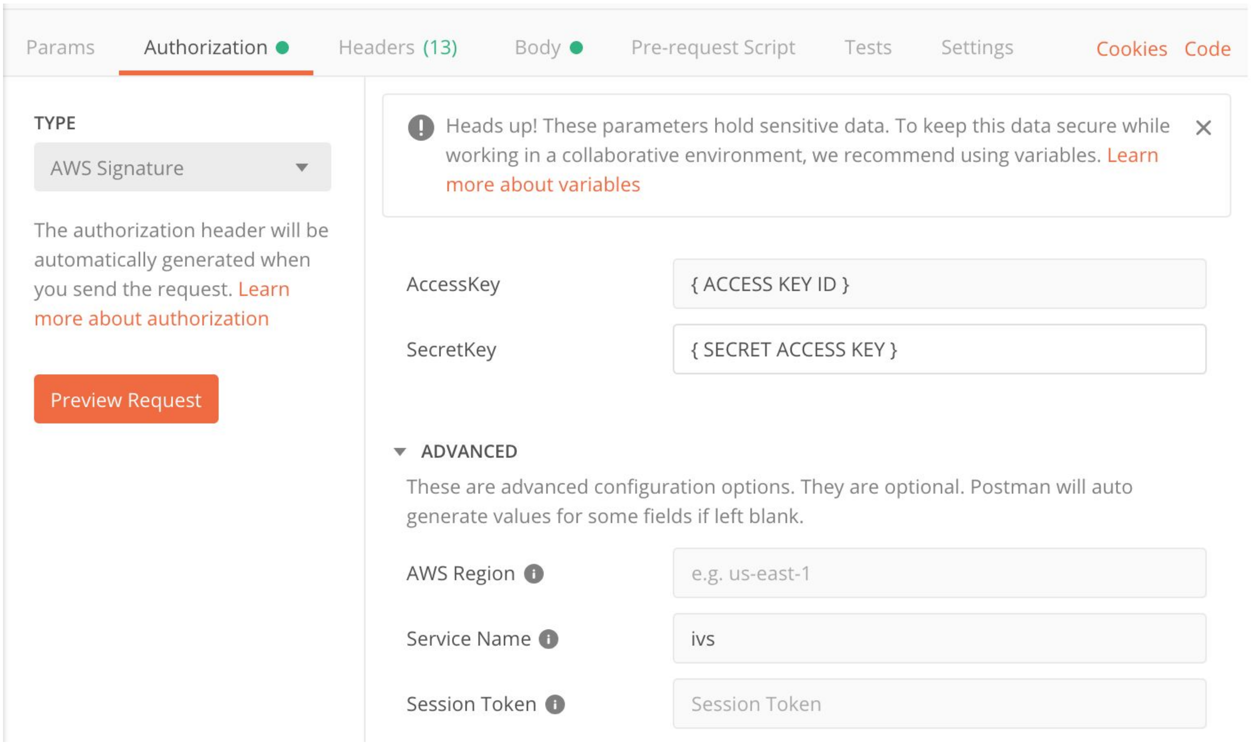
Timed Metadata is metadata that is bound to a certain timestamp on your live video feed. When you call the *PutMetadata* endpoint, you supply a string of information that will be delivered at the nearest upcoming timestamp in the stream. Rather than all viewers receiving this information at the moment you send it, viewers will instead receive this information when they consume that specific timestamp in your livestream. This way, you can ensure that certain bits of metadata will not be received until your viewers have passed a certain point in your video feed.

We can test our ability to insert Timed Metadata into our livestream using Postman. Create a new Postman request and configure the following:

First, determine the AWS Region your channel is in by looking at your IVS Channel ARN, which should be in the format of *arn:aws:ivs:{ AWS_REGION }:etc*. Set the Postman request URL to whichever Amazon IVS service endpoint corresponds to the region your channel is in, and target the /PutMetadata route. The request type will be POST.


POST ivs.us-east-1.amazonaws.com/PutMetadata

Select the Authorization tab and change your Authorization Type to AWS Signature. Paste the Access Key ID and Secret Access Key of your IAM User into their corresponding fields. Under Advanced, for Service Name, put *ivs*.


Params | Authorization ● | Headers (13) | Body ● | Pre-request Script | Tests | Settings | Cookies | Code

TYPE

AWS Signature

The authorization header will be automatically generated when you send the request. Learn more about authorization

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

AccessKey: { ACCESS KEY ID }

SecretKey: { SECRET ACCESS KEY }

ADVANCED

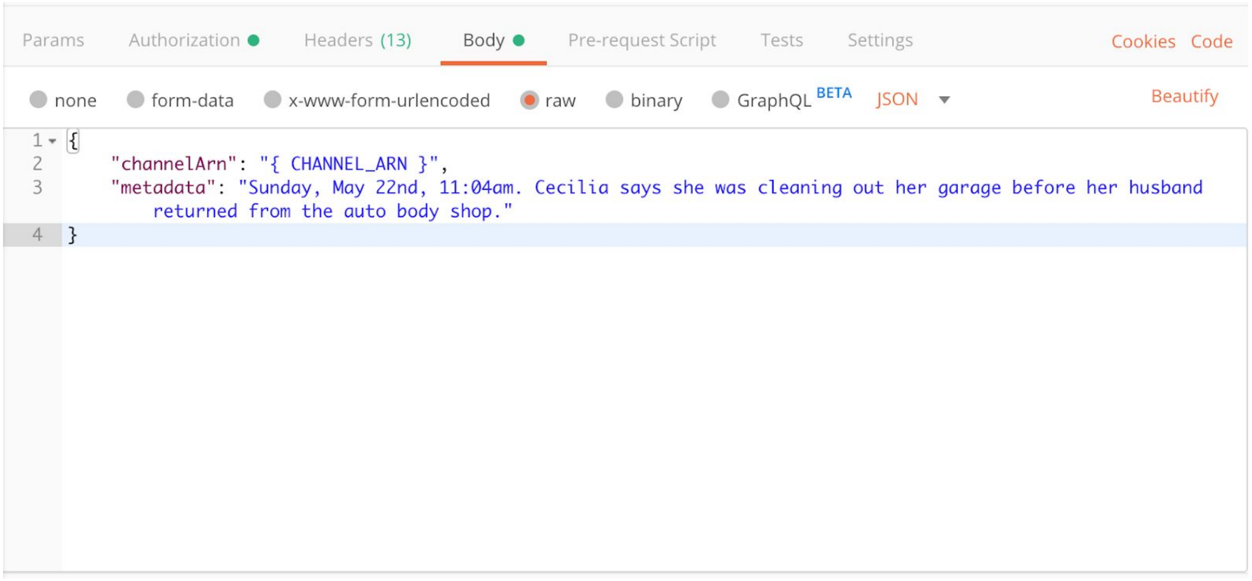
These are advanced configuration options. They are optional. Postman will auto generate values for some fields if left blank.

AWS Region: e.g. us-east-1

Service Name: ivs

Session Token: Session Token

Now set your request body as raw JSON in accordance with the [PutMetadata API documentation](). This is our chance to send a clue for our murder mystery as the *metadata* sent in this request.


Params | Authorization ● | Headers (13) | Body ● | Pre-request Script | Tests | Settings | Cookies | Code

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL BETA | JSON | Beautify

```
{
    "channelArn": "{ CHANNEL_ARN }",
    "metadata": "Sunday, May 22nd, 11:04am. Cecilia says she was cleaning out her garage before her husband
        returned from the auto body shop."
}
```

We need a way to represent all of our murder mystery clues within *custom-player-controls/script.js*. Let's define them as an empty, global array at the top of the file for now.

```
const clues = [];
```

Our boilerplate already provides an anonymous callback function for handling incoming Timed Metadata. By modifying the callback function defined in the EventListener for the PlayerEventType.TEXT_METADATA_CUE, we can take the metadata text and push it to our list of clues. Then, we can render our list of clues to our *notepad*.

```
player.addEventListener(PlayerEventType.TEXT_METADATA_CUE, function (cue) {
    const clue = cue.text;
    clues.push(clue);
    document.getElementById("notepad").value = clues.join("\n");
});
```

Now, when we fire our PutMetadata API request via Postman while our livestream is running, our web app will pull the clue from the supplied metadata and insert it directly onto our notepad!

Next Steps

With this, the basic functionality of our interactive murder mystery app is complete. There are a lot of ways to extend this project. You can:

- Create a companion application that allows a broadcaster to create clues in advance and send them directly to the AWS PutMetadata endpoint.
- Create a more robust notepad that parses metadata clues as JSON and separates clues into categories such as suspects, motives, etc.
- Create a submission system where a viewer can try to solve the mystery once the stream ends. Who will be the first to put the pieces together?

I hope you found this tutorial helpful! If you have any feedback or questions for the author, feel free to message me on Twitter @berniemarger. I would love to hear what you're building!

See you next time 👋