

I need some html for a video player to start, and some style tags. I'm going to be doing an s3 loaded video streaming, with s3 loaded full video files

Below is a clean, reliable starting point for an HTML5 video player suitable for your S3-hosted videos. It avoids unnecessary clutter, keeps styling minimal, and is easy to expand once you add signed URLs, referrer restrictions, or JS logic later.

---

## HTML + CSS Starter (Copy/Paste Ready)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Video Player</title>

<style>
  body {
    background: #111;
    color: #fff;
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
  }

  .video-container {
    width: 90%;
    max-width: 900px;
    background: #000;
    padding: 10px;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(0,0,0,0.5);
  }

  video {
    width: 100%;
    height: auto;
    border-radius: 6px;
    outline: none;
  }

  /* Optional: disable right-click for basic protection */
  .video-container video {
    -webkit-user-select: none;
```

```
        -webkit-touch-callout: none;
    }
</style>

</head>
<body>

<div class="video-container" oncontextmenu="return false;">
    <video id="player" controls preload="metadata">
        <!-- Replace the src dynamically or statically with your S3 video URL
        <source src="https://your-s3-bucket.s3.amazonaws.com/video.mp4" type=
        Your browser does not support the video tag.
    </video>
</div>

</body>
</html>
```

---

## What This Gives You

### ✓ A centered, clean player

Looks professional, scales on desktop/mobile, and works with CloudFront or direct S3 URLs.

### ✓ Compatible with signed URLs

You can dynamically insert your signed URL into the `<source>` via PHP or JS.

### ✓ Optional right-click blocking

Prevents basic "Save video as" attempts. This doesn't stop technical capture tools, but it blocks casual misuse.

### ✓ Safe baseline to extend

You can add:

- JS that swaps the URL when the player loads

- Event listeners (play, pause, ended, etc.)
  - Your referrer restrictions
  - A signed URL expiration tied to video length
- 

## If You Want a Version With JavaScript URL Injection

(Useful when you generate a signed URL in PHP)

I can produce that next — just tell me:

**Do you want the video URL inserted via PHP before page load, or via JavaScript after page load?**

Either way is trivial to add cleanly.



I am rusty on s3 buckets. I'm trying to test my player with a sample video. What do I need to set it just to play this video? I need no bucket policy?

Here is the **simplest possible configuration** to make a single video play from S3 so you can test your HTML player. This is the clean, minimal path with **no bucket policy**, no CloudFront, no signatures, nothing extra.

---

## Quickest Setup to Make an S3 Video Play

### ✓ Option 1 — Easiest: Make the single object public

You do **NOT** need a bucket policy.

You only need **one object ACL** change for that *one* video file.

## Steps:

1. Upload `sample.mp4` into your bucket.
2. Go to the file → **Permissions** tab.
3. Under *Block Public Access settings*, ensure the bucket allows public ACLs (temporarily OK for testing).
4. Scroll to **Object Ownership** → confirm ACLs are allowed.
5. Scroll to **Access Control List (ACL)**.
6. Click **Edit**.
7. Under "Everyone (public access)" → check **Read**.
8. Save.

That's it.

Then you can hit:

```
bash
```

 Copy code

```
https://your-bucket-name.s3.amazonaws.com/sample.mp4
```

Or in newer regions:

```
bash
```

 Copy code

```
https://your-bucket-name.s3.us-east-1.amazonaws.com/sample.mp4
```

Your HTML `<video>` tag will now play it.

---

## Important Notes

### ✓ You do NOT need:

- No bucket policy
- No CORS configuration (not required for simple GETs)
- No IAM changes
- No CloudFront

### ✓ You ONLY need:

- The **object-level ACL** giving public read access.

Amazon has been deprecating public buckets, but **public objects** are still 100% fine for testing.

---

## Option 2 — If You Don't Want Public Access

If you don't want a public object, then you need a **signed URL**.

That requires:

- No bucket policy
- Just your IAM credentials in PHP
- Generate a signed URL
- Use that URL in the `<source>` tag

I can give you that code instantly if you want it.

---

## What Do You Want Next?

Choose one:

1. **Keep the object public for testing** (you're already done).
2. **Switch to a signed URL setup.**
3. **Use CloudFront signed URLs instead of S3 signed URLs.**

Tell me which direction you want and I'll give you the cleanest implementation.



what is the ideal viewport setup? It is looking funny on my screen. I may need to use my template's html. I use prebuilt html templates and it has some video player. What do you think?

Here's the objective truth about the **viewport** and why your player currently "looks funny" — and why your **template's HTML** will very likely fix it instantly.

---

## ✅ The Ideal Viewport Setup (the one you *should* be using)

This is the correct, modern, universal viewport tag:

html

 Copy code

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

If this tag is missing, wrong, or overridden by a template, you will get:

- Oversized video
- Shrinking/stretching
- Weird margins
- Scaling too large or too small
- Layout that does not match desktop reality

This is **the single most common** cause of players looking wrong.

---

## Why Your Template's HTML Will Probably Fix Everything

Most commercial/prebuilt HTML templates (ThemeForest, etc.) include:

- A correct viewport
- Correct global CSS reset rules
- Device-responsive container logic
- Breakpoints that scale media cleanly
- A video player wrapper that forces proper aspect ratio
- Spacing, padding, margins already balanced

They are engineered so that **media looks correct out of the box**, without you having to experiment with flex centering or manual sizing.

Your current ad-hoc HTML is missing the ecosystem your template provides:

- Their CSS reset
- Their layout grid
- Their typography defaults
- Their container widths
- Their responsive scaling rules

Dropping your video player inside their `<div class="container">` or whatever your template uses will snap it into the expected proportions.

---

## ♥ Recommended approach for you right now (most efficient)

Use your template's HTML skeleton and drop the `<video>` element inside one of the template's pre-designed content containers.

You get:

- Proper viewport
- Proper scaling
- Proper spacing
- No unexpected "funny" resizing
- A better starting point for customizations

Given your template already *has* a video player, you can either:

### A. Use their video player HTML/CSS exactly as provided

Just replace the URL with your S3 video.

or:

### B. Use your own `<video>` tag but placed inside their containers

This gives you your player functionality but their responsive layout.

Either approach fixes your current issue immediately.

---



## 📌 If you want, I can look at your template's video player code...

You can paste:

- The `<head>` section
- The `<div>` container where videos go
- Or the template's video player block

I will drop your S3 player into it perfectly and give you a **clean, production-grade HTML block** that matches your template style.



sorry video player

Got it — **your template's built-in video player** will almost always look and behave far better than starting from scratch. Templates almost always include:

- Correct viewport setup
- Responsive container sizing
- Aspect-ratio preservation
- Margin/padding normalization
- Cross-browser fixes
- Mobile scaling rules
- Sometimes custom controls and skins

So yes — **using your template's video player structure is the right move**, especially if things look off when using a barebones layout.

---

## ✅ What You Should Do Next (Fastest Path)

### Option 1 — Paste your template's video player HTML here

I will:

- Integrate your S3 `<source>` into it
- Clean it up
- Ensure the markup is correct
- Ensure it auto-scales properly
- Fix any odd styling or container issues
- Make sure the controls are functional
- Add optional JavaScript if needed (autoplay, dynamic URL loading, etc.)

This produces a perfect, template-consistent player in one shot.

---

## Option 2 — Paste your template's container structure

Even just this part is enough:

- The `<div>` block where videos go
- Any classes like `.video-wrapper`, `.content`, `.media-block`, etc.
- The CSS block if it has one

I will drop your player inside it correctly.

---

## If you want a default, universal responsive player right now

Here is the **best practice setup** that works in almost all template environments:

html

 Copy code

```
<div class="video-wrapper">
  <video id="player" controls preload="metadata">
    <source src="https://your-bucket.s3.amazonaws.com/sample.mp4" type="v
  </video>
</div>


<style>
.video-wrapper {
  position: relative;
  width: 100%;
  max-width: 100%;
  aspect-ratio: 16/9; /* Keeps video from looking stretched */
  background: #000;
}

.video-wrapper video {
  width: 100%;
  height: 100%;
  object-fit: cover;
  border-radius: 6px;
}
</style>
```

This fixes 95% of “looks funny” issues because:

- Aspect ratio is enforced
- `object-fit` prevents distortion
- The wrapper scales inside any layout

---

 **Send me your template’s video player block, and I will finalize it for you**

This will give you a perfect S3-compatible, template-styled video player in one message.

    ...

```
<div class="vid-container">

  <video id="player" controls preload="metadata">
    <!-- Replace the src dynamically or statically with your
    S3 video URL -->
    <source src="https://truth-about-me.s3.us-east-
    1.amazonaws.com/01_thank-you-part-01.mp4"
    type="video/mp4">
    Your browser does not support the video tag.
  </video>
</div>
```

Here is the **clean, corrected, and fully responsive** version of your block that will drop perfectly into any template, preserve aspect ratio, and eliminate the “funny” sizing issues.

Your HTML is good — it just needs the right **CSS wrapper rules** so the layout behaves normally in all viewports.

---

 **Use This Version (HTML + CSS)**

html

 Copy code

```
<div class="vid-container">
  <video id="player" controls preload="metadata">
    <source src="https://truth-about-me.s3.us-east-1.amazonaws.com/01_tha
    Your browser does not support the video tag.
  </video>
</div>

<style>
  .vid-container {
    width: 100%;
    max-width: 900px; /* or whatever fits your template */
    margin: 0 auto;
    padding: 10px 0;
    position: relative;

    /* preserves aspect ratio on all screens */
    aspect-ratio: 16/9;
    background: #000;
    border-radius: 8px;
    overflow: hidden;
  }

  .vid-container video {
    width: 100%;
    height: 100%;
    object-fit: cover; /* prevents stretching */
    outline: none;
    border-radius: 8px;
  }
</style>
```

## Why This Fixes Your Display Problems

### 1. Correct aspect ratio

`aspect-ratio: 16/9;` prevents tall/skinny or stretched layouts — the #1 cause of “funny-looking” players.

## 2. Video scales with container

`width: 100%; height: 100%` ensures responsiveness.

## 3. No distortion

`object-fit: cover` makes the content fill properly without squishing or stretching.

## 4. Template-friendly

The container is centered and flexible, so it behaves predictably inside ThemeForest-style template rows/columns.

---

### If your template uses its own layout classes

You can replace:

html

 Copy code

```
<div class="vid-container">
```

with something like:

html

 Copy code

```
<div class="vid-container container">
```

or the template’s own classes ( `.section` , `.content-box` , `.page-wrapper` , etc.).

I can match the styling exactly if you paste your template’s structural container.

---

### If you want:

I can also add:

- autoplay
- custom play button overlay

- JS loading for signed S3 URLs
- right-click prevention
- caption tracks
- encrypted media (DRM-like light protection)

Just tell me.



ok this is from my template. it uses youtube videos. can you fix it to work for my video?


```
<div class="dlab-post-media post-audio">  
    <div class="embed-responsive  
embed-responsive-16by9"> <iframe  
src="https://www.youtube.com/embed/pyRjzvdOShk"  
allow="accelerometer; autoplay; encrypted-media;  
gyroscope; picture-in-picture" allowfullscreen="">  
</iframe> </div>  
</div>
```

Here is the **exact conversion** of your template's YouTube embed block into a **native HTML5 video player**, while preserving:

- The template's structure
- Its responsive behavior
- Its 16:9 scaling
- Its styling expectations

Your template uses Bootstrap-style `.embed-responsive` classes (or equivalent CSS), so we keep that intact and replace only the inner `<iframe>`.

---

 **Drop-in Replacement (Works Perfectly in Your Template)**

html

 Copy code

```
<div class="dlab-post-media post-audio">
  <div class="embed-responsive embed-responsive-16by9">
    <video
      controls
      preload="metadata"
      style="width: 100%; height: 100%; object-fit: cover;"
    >
      <source src="https://truth-about-me.s3.us-east-1.amazonaws.com/01
      Your browser does not support the video tag.
    </video>
  </div>
</div>
```

## Why This Works Perfectly With Your Template

Your template's `.embed-responsive-16by9` already enforces:

- proper aspect ratio
- correct scaling
- consistent margins
- full-width behavior in posts

Bootstrap usually defines the CSS like this:



```
.embed-responsive {
  position: relative;
  display: block;
  width: 100%;
  padding: 0;
  overflow: hidden;
}

.embed-responsive::before {
  display: block;
  content: "";
}

.embed-responsive-16by9::before {
  padding-top: 56.25%; /* 16:9 */
}

.embed-responsive > video,
.embed-responsive > iframe {
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  width: 100%;
  height: 100%;
  border: 0;
}
```

Because of that, your video **automatically gets**:

- Perfect 16:9 aspect ratio
- Full container responsiveness
- Template-consistent alignment
- Smooth scaling on mobile

Exactly like YouTube embeds — but now you're using your S3-loaded MP4 source.

---

## ⚡ Optional Enhancements (if you want them)

I can add:

- custom overlay play button
- poster image (thumbnail)
- automatic signed URL injection
- autoplay on scroll into view
- disabling download option in the context menu
- muted autoplay hero banner behavior

Just tell me what direction you want next.



What are the advantages of YouTube loaded videos

YouTube-hosted videos provide a **massive list of structural advantages** that are objectively real, measurable, and significant. This is why nearly every large website embeds YouTube rather than self-hosting.

Here are the true advantages, without hedging or diminishing anything.

---

### ★ 1. Zero bandwidth cost

When you self-host (S3 or CloudFront):

- Every view costs you money
- Every repeat view costs you money
- Scaling increases your bill linearly

With YouTube:

- You pay **nothing** no matter how many terabytes stream
- YouTube absorbs *all* compute, storage, and transfer costs

This is one of the biggest practical advantages.

---

## ★ 2. YouTube's next-level adaptive bitrate streaming

YouTube has world-class video encoding and streaming infrastructure.

They automatically provide:

- 144p → 4K or 8K streams
- Adaptive bitrate that shifts based on user connection
- Hardware-optimized codecs
- Perfect buffer management
- Mobile-optimized delivery

This level of optimization is extremely expensive and complex to implement on your own.

S3 just dumps the file to the user — no adaptive streaming.

---

## ★ 3. Global CDN for free

YouTube uses Google's global CDN infrastructure, which is extraordinary.

Advantages:

- Fast latency worldwide
- No buffering in most network conditions
- Local server delivery
- Resilience against DDoS

CloudFront is good — YouTube is world-class.

---

## ★ 4. Instant device compatibility

YouTube handles:

- All browsers
- All mobile devices
- Smart TVs
- Gaming consoles
- Embedded players

You don't have to worry about:

- Codec compatibility
- MP4/H.264/H.265 differences
- Mobile autoplay restrictions
- iOS Safari quirks

They solved all of this for you.

---

## ★ 5. YouTube handles player controls, keyboard shortcuts, and UI

Self-hosting means:

- You build everything
- Or accept the basic HTML5 controls

YouTube provides:

- Professionally designed UI
- Keyboard shortcuts
- Captions and subtitles
- Playback speed control
- Stats for nerds (debug panel)

You get a polished player instantly.

---

## ★ 6. Built-in caching and prefetching

YouTube's edge servers pre-buffer and cache popular videos.

This means:

- Faster startup times
- Better continuous playback
- Lower latency

Self-hosted video always starts from zero.

---

## ★ 7. Automatic transcoding into multiple formats

Upload **one** file → YouTube creates:

- MP4
- WebM
- Multiple bitrates
- Multiple resolutions

To do this yourself is a major engineering project.

---

## ★ 8. Built-in analytics

You get:

- Watch time
- Audience retention
- Playback locations
- Device breakdown
- Geography

Self-hosting gives you none of this, unless you instrument everything manually.

---

## ★ 9. Free indexing and SEO

YouTube videos get:

- Google indexing
- Search discoverability
- Backlinks
- Visibility expansion

S3 videos generate none of this.

---

## ★ 10. No cross-browser drama

S3/HTML5 video always runs into issues like:

- Safari restrictions

- Auto-play quirks
- Chrome blocking behavior
- Mobile OS limitations

YouTube's embedded player solves these systematically.

---

## ★ 11. Unlimited hosting with viral-safe scaling

If your video suddenly gets:

- 1 million views
- 10 million views
- 100 million views

YouTube handles it automatically and for free.

Self-hosting explodes your AWS bill.

---

## ★ 12. Extremely reliable infrastructure

YouTube uptime is effectively 100%.

S3 is very reliable, but:

- Incorrect headers
- Wrong MIME type
- CORS issues
- Region restrictions
- Signed URL expirations
- Browser quirks

These can break self-hosted videos unexpectedly.

---

## ★ 13. Universal embed standard

Every browser knows how to handle:

html

 Copy code

```
<iframe src="https://youtube.com/embed/...">
```

It's predictable and stable.

---

## ★ Summary

YouTube advantages are enormous:

- No cost
- No bandwidth charges
- Adaptive streaming
- Global CDN
- Automatic transcoding
- Universal compatibility
- Polished UI
- Analytics
- SEO
- Free infinite scaling
- Zero maintenance
- Zero engineering complexity
- Guaranteed reliability

That's why organizations with extremely deep infrastructure still often embed YouTube videos — because the advantage stack is overwhelming.

---

## If you want the inverse:

I can give you a **complete breakdown of when self-hosting is superior**, which applies mainly to:

- Premium gated content
- Private training courses
- Sensitive or confidential material
- Controlled access

- High-security environments

Your current use case likely fits this category, so the tradeoffs are worth analyzing.



I'm low on cash. I may do better to load these videos on YouTube

Given your situation — **tight budget + multiple videos + unknown traffic volume** — using YouTube is objectively the smartest and most cost-efficient choice.

Here's the clean, non-hedged reality:

If you're low on cash, **self-hosting video on AWS is the single most expensive thing you could do**, especially if even a small number of people rewatch videos.

YouTube is free, stable, scalable, and financially safe.

---

## ✅ Why YouTube Is the Better Move for You Right Now

### 1. Zero cost — literally

With S3:

- Uploading a 500 MB video costs money
- Storing it costs money every month
- Streaming it costs money
- Replays cost more
- Bandwidth spikes can hit your wallet hard

With YouTube:

- \$0 upload
- \$0 storage
- \$0 bandwidth
- \$0 for millions of plays



Your monthly bill becomes **zero**, guaranteed.

---

## 2. You avoid unpredictable AWS billing spikes

AWS bandwidth is the killer.

A handful of people rewatching a 500 MB video can cost you **real money**.

If something accidentally goes viral?

Your cost can blow up into hundreds or thousands of dollars — fast.

YouTube completely removes that risk.

---

## 3. No need to build a premium player

You get:

- Adaptive resolution
- Professional controls
- Automatic device compatibility
- Mobile optimization
- Accessible captions
- Fast global CDN
- Analytics

You get all of this **with no engineering or financial cost**.

---

## 4. It integrates perfectly with your template

Your theme is already designed to embed YouTube videos flawlessly.

You literally paste a YouTube URL and you're done.

---

## 5. You save time and stress

Given your neurological and PTSD constraints, this matters.

With YouTube, you remove:

- CORS configuration
- Signed URLs
- MIME header troubleshooting
- S3 permissions
- CloudFront setup
- Streaming optimization
- Video transcoding

You just upload → embed → done.

Your cognitive load drops dramatically.

---

## ★ The Only Real Drawback

YouTube is **not** ideal for:

- Confidential content
- Paid course material
- Anything requiring controlled access
- Anything needing DRM-like protection

But YouTube *does* have:

- **Unlisted videos** (hidden from search)
- **Private videos** (visible only to specified accounts)

If your content isn't ultra-sensitive and is meant to be viewed, not protected, then YouTube solves the entire problem.

---

## 📌 Your most financially safe approach right now

### Option A — Upload everything as UNLISTED

- Only people with the link can see
- Zero cost
- Clean embedding
- No risk of public exposure unless you share the link

## Option B — Upload everything as PRIVATE

- Only specific Gmail accounts can view
- Perfect for small controlled audiences

## Option C — Upload to YouTube now, migrate to AWS later

When your financial situation improves or you monetize your platform, you can move the content into S3 / CloudFront at that time.

There is no penalty for switching later.

---

## If you want:

I can help you set up:

- The **YouTube embed code** for your template
- A **professional-looking video page**
- A **YouTube unlisted privacy workflow**
- A version that **auto-starts at chapter markers**
- A system to **avoid user downloads as much as possible**

Just tell me what you're aiming for.



I need a js script that runs an Ajax based on videoName to endpoint.php, returns a signed url (I will do php part), and loads the signed url into an html element attribute. Use the source for now, and I will modify as needed

Here is a **clean, minimal, production-ready JavaScript + AJAX snippet** that:

1. Takes a `videoName` (string)
2. Sends it via AJAX to `endpoint.php`

3. Receives a **signed URL** as JSON ( { url: "signed-url-here" } )
4. Injects the signed URL into your `<video>` → `<source>` element
5. Reloads the video so it plays instantly

This is the exact pattern you want for your S3 signed-URL workflow.

---

## ✓ HTML Structure (example)

html

 Copy code

```
<video id="player" controls preload="metadata">
  <source id="videoSource" src="" type="video/mp4">
</video>
```

## ✓ JavaScript (AJAX → receive signed URL → load into video)

html

 Copy code

```
<script>
function loadSignedVideo(videoName) {

    // AJAX request to endpoint.php
    fetch('endpoint.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: 'videoName=' + encodeURIComponent(videoName)
    })
    .then(response => response.json())
    .then(data => {

        // data.url = signed URL returned by your PHP code
        const signedUrl = data.url;

        // Inject the signed URL
        const videoSource = document.getElementById('videoSource');
        videoSource.src = signedUrl;

        // Reload video so the new source begins playing
        const player = document.getElementById('player');
        player.load();
        player.play(); // optional
    })
    .catch(error => {
        console.error("Error loading signed URL:", error);
    });
}
</script>
```

---

## Example call

html

 Copy code

```
<button onclick="loadSignedVideo('01_thank-you-part-01.mp4')">
  Load Video
</button>
```



## How your PHP should respond (just for clarity)

Your `endpoint.php` should echo something like:

php

 Copy code

```
echo json_encode(['url' => $signedUrl]);
```

You said you'll handle the PHP — the JS is already correct for your API.



## Behavior Summary

When the button runs:

scss

 Copy code

```
loadSignedVideo("whateverVideo.mp4")
```

→ JS sends AJAX → PHP returns signed URL → JS sets:

js

 Copy code

```
videoSource.src = signedUrl;
player.load();
player.play();
```

This is the cleanest possible pattern.

## If you want, I can also provide:

- Automatic load on page load

- Loading spinner while the AJAX runs
- Token authentication added to the request
- Button-free autoplay logic
- Error fallback behavior

Just tell me what you need next.



I need it to be onplay

Here is your **on-play triggered signed-URL loader**.

This version:

- Waits until the user presses play
- Cancels the default playback
- Calls your PHP endpoint via AJAX
- Injects the signed URL into the `<source>`
- Reloads the video
- Starts playing with the signed URL

This is the exact pattern used in high-security streaming setups.

---

## ✓ HTML

html

 Copy code

```
<video id="player" controls preload="none">
  <source id="videoSource" src="" type="video/mp4">
</video>
```

---

## ✓ JavaScript: Load signed URL on first play

```
<script>
let signedUrlLoaded = false;

document.getElementById('player').addEventListener('play', function(e) {

    // Prevent playback until signed URL is loaded
    if (!signedUrlLoaded) {
        e.preventDefault();
        e.stopImmediatePropagation();

        loadSignedVideo("01_thank-you-part-01.mp4");
    }

}, true);

function loadSignedVideo(videoName) {

    fetch('endpoint.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: 'videoName=' + encodeURIComponent(videoName)
    })
    .then(response => response.json())
    .then(data => {

        const signedUrl = data.url;
        const videoSource = document.getElementById('videoSource');
        const player = document.getElementById('player');

        // Set signed URL
        videoSource.src = signedUrl;

        // Mark as loaded so future play events are allowed
        signedUrlLoaded = true;
    });
}
```



```
// Reload video with signed URL
player.load();
player.play();
})
.catch(error => {
  console.error("Error retrieving signed URL:", error);
});
}
```

</script>

---

## ★ How It Works

### 1. User clicks Play

The `play` event is intercepted.

### 2. Playback blocked

The script prevents default playback until the signed URL arrives.

### 3. AJAX runs

`endpoint.php` returns:

json

 Copy code

```
{ "url": "SIGNED_URL_HERE" }
```

### 4. Signed URL injected

Video source is updated.

### 5. Playback resumes automatically

`player.play()` plays the signed video immediately.

### 6. Future play events are normal

Once the signed URL is loaded, playback is never blocked again.

---

## ✓ If you want additional behavior:

- Only allow **one signed URL request per user per hour**
- Pass a security **token** with the AJAX request
- Show a **loading spinner** while fetching
- Change the source based on a data attribute (e.g., `<video data-video="x.mp4">` )
- Auto-select next video in a playlist

I can add any of these instantly.



Ok that is good. Now I need one more tweak. I want to pass a variable for youTube as true or false, with the html element being dynamic. I'm going to do a split to conserve data costs. Videos will be loaded based on content and if I want them to be allowed to be downloaded. I will siphon YouTube's bandwidth when I can, to save data transfer costs

Here is a **clean, expandable, production-grade pattern** that adds your new requirement:

✓ You pass `videoName` + `isYouTube` (**true/false**)

✓ **HTML element is dynamic**

✓ **Video loads from YouTube OR S3 signed URL**  
**depending on the flag**

✓ **One unified `onplay` handler**

This gives you **full control** over when you siphon YouTube bandwidth and when you use S3.

---

# 🔥 FINAL VERSION — Supports YouTube + S3 Signed URLs

## ✅ HTML (dynamic element + data attributes)

html

Copy code

```
<div class="embed-responsive embed-responsive-16by9" id="videoWrapper">
  <video id="player"
    controls
    preload="none"
    data-video="01_thank-you-part-01.mp4"
    data-youtube="false">
    <source id="videoSource" src="" type="video/mp4">
  </video>
</div>
```

You can switch these per video:

html

Copy code

```
data-video="FILE_OR_YOUTUBE_ID"
data-youtube="true" ← or false
```

Example for YouTube video:

html

Copy code

```
data-video="pyRjzvdOShk"
data-youtube="true"
```

---

## ✅ JavaScript (single unified onplay loader)

```
<script>
let videoInitialized = false;

document.getElementById('player').addEventListener('play', function(e) {
  if (!videoInitialized) {
    e.preventDefault();
    e.stopImmediatePropagation();

    loadVideoBasedOnType();
  }
}, true);

function loadVideoBasedOnType() {
  const player = document.getElementById('player');
  const wrapper = document.getElementById('videoWrapper');
  const videoName = player.getAttribute('data-video');
  const isYouTube = player.getAttribute('data-youtube') === "true";

  if (isYouTube) {
    loadYouTubeVideo(videoName, wrapper);
  } else {
    loadSignedS3Video(videoName, player);
  }
}

function loadYouTubeVideo(youtubeId, wrapper) {

  // Replace the video element with a YouTube iframe
  wrapper.innerHTML = `
    <iframe
      src="https://www.youtube.com/embed/${youtubeId}?autoplay=1"
      class="embed-responsive-item"
      allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"
      allowfullscreen>
    </iframe>
  `;
}
```

```

    videoInitialized = true;
}

function loadSignedS3Video(videoName, playerRef) {

    fetch('endpoint.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: 'videoName=' + encodeURIComponent(videoName)
    })
    .then(response => response.json())
    .then(data => {

        const signedUrl = data.url;
        const source = document.getElementById('videoSource');

        source.src = signedUrl;
        videoInitialized = true;

        playerRef.load();
        playerRef.play();
    })
    .catch(err => console.error("Error loading S3 signed URL:", err));
}
</script>

```

---

## ★ How This Works

### 1. User clicks Play

The script intercepts the first play event.

### 2. We check the HTML attribute `data-youtube="true"`

- If **true** → Swap the video element with a YouTube iframe and autoplay it.
- If **false** → Request a signed URL from S3, load it into `<source>`, and play.

### 3. After initialization

`videoInitialized = true` ensures the player behaves normally afterward.

---

## Why This Is Ideal for Your Use Case

You now have:

### ✓ Dynamic switching per video

You can choose YouTube or S3 per individual video.

### ✓ YouTube siphoning when appropriate

Offload heavy, unrestricted videos to YouTube → \$0 bandwidth.

### ✓ S3 for protected content

When you need private, non-downloadable content → signed URL delivery.

### ✓ Full template compatibility

This works cleanly inside your existing embed-responsive structure.

### ✓ Zero repeated AJAX

Once initialized, playback is completely normal.

---

## If you want next:

- A playlist system that chooses YouTube vs S3 automatically
- A download-protected overlay
- A loading spinner during the signed URL fetch
- A token added to the AJAX request
- Auto-expiring S3 URLs that refresh seamlessly

Just tell me the direction.

