How to Use OBS Studio to Connect to a Livestream on Server 2b

To connect to and view a livestream hosted on Server 2b. Here's how:

Steps to Connect to Livestream

1. Install OBS Studio

- Download OBS Studio from the official website: https://obsproject.com/.
- Install the appropriate version for your operating system (Windows, macOS, Linux).

2. Launch OBS Studio

Open the OBS Studio application after installation.

3. Set Up a New Source

- In OBS Studio, locate the **Sources** box at the bottom of the interface.
- Click the + **(Add)** button to add a new source.

4. Add a Media Source

- From the popup list, select **Media Source** and click **OK**.
- Name the source (e.g., Livestream 2b) and click **OK**.

5. Configure the Stream URL

- In the **Media Source** settings window:
- Check **Input URL**.
- Enter the livestream URL from **Server 2b**.
  Example:

http://<*server-ip*>:8000/live/stream_key_2a

- Enable the option **Use hardware decoding if available** if your system supports it.
- Click **OK**.

6. Adjust the OBS Canvas

- Resize the preview window in OBS Studio to fit the livestream appropriately.
- Drag and resize the Media Source in the **Preview Window** area as necessary.

7. Start Streaming or Preview

- If you intend to **stream**, click **Start Streaming** in the OBS control panel.
- If you're just viewing the stream, use **Start Recording** or monitor the stream directly in OBS.

Troubleshooting

1. **Livestream Not Visible in OBS**:

- Double-check the livestream URL.
- Ensure the server is running and accessible.

2. **OBS Doesn't Detect Stream**:

- Verify the port or URL is correct.
- Check firewall settings to ensure the necessary port is open.

3. **Playback Lag**:

- Enable or disable **Use hardware decoding if available** in the Media Source settings.
- Monitor your system performance (too many background tasks may interfere).

How to Use VideoLAN (VLC) to Connect to a Livestream on Server 2b

To connect to a livestream hosted on Server 2b, follow the steps below:

Steps to Connect to Livestream

1. Install VideoLAN (VLC) Media Player

- Download VLC Media Player from https://www.videolan.org/vlc/.
- Install the appropriate version for your operating system (Windows, macOS, Linux).

2. Launch VLC Media Player

Open VLC Media Player after installation.

3. Open the Stream URL

- Go to the menu bar at the top of VLC:
- **Windows/Linux**: Media → Open Network Stream
- **macOS**: File → Open Network
- Alternatively, press Ctrl + N (Windows/Linux) or Cmd + N (macOS).

4. Enter the Stream URL

You'll need the livestream URL from **Server 2b**. This should follow the WebRTC endpoint or stream protocol. For example:

http://*SERVER_2B_IP:PORT/stream*

Example Stream URL:
If Server 2b streams to port 8000, your URL might look like:

http://*<server-ip>:8000/live/stream_key_2a*

Paste the URL into the **Network URL box**.

5. Start Playback

Click the **Play** button after entering the URL. VideoLAN will connect to the stream and begin playback.

Troubleshooting

1. **Stream Not Found**:

• Ensure Server 2b is running and accessible from your network.

• Check firewall rules to ensure port forwarding for the stream is open.

2. **Playback Issues**:

• Confirm that the URL is correct.

• Restart VideoLAN and retry the connection.

3. **Port/Connection Blocked**:

• Ensure port 8000 (or the respective port used by Server 2b) is open in any router or firewall.

Using **VLC** with VideoLAN's built-in streaming support makes monitoring live streams or live overlays simple and effective. If you experience any other issues connecting, check the live server's logs or confirm the signaling server configuration.

1. Docker Compose File: docker-compose.yml

This is the main Docker Compose configuration file used to orchestrate multiple services involved in live streaming, recording, WebRTC signaling, and live overlays.

Services Defined in the Compose File

1. live-streaming-server-a

- **Image:** tiangolo/nginx-rtmp
- **Purpose:** Acts as a live-streaming server with RTMP support.

- **Ports:**
- 1935:1935: Maps the RTMP streaming port.
- 8080:80: Maps an optional HTTP server for monitoring.
- 8081:8080: Additional HTTP mapping.
- **Environment Variables:** Configured using .env.
- **Networking Configuration:** Connected to live_stream_network with a specific IP address.
- **Command Customization:** Dynamically configures RTSP streams with a bash script to pull RTSP camera streams dynamically into the server.

2. live-streaming-server-b

- **Image:** tiangolo/nginx-rtmp
- **Purpose:** A second RTMP live-streaming server for a specific segment or another live streaming purpose.

- **Ports:**
- 1936:1935
- 8083:80
- 8084:8080
- **Networking Configuration:** Also connected to live_stream_network.
- **Command Customization:** Installs GStreamer and executes stream.sh, which handles streams.

3. live-recording-server

- **Image:** jrottenberg/ffmpeg:4.4-ubuntu
- **Purpose:** Responsible for recording live streams.
- **Ports:**
- 1937:1935
- 8084:8080

- **Environment Variables:** Reads streaming server details to manage recording streams.
- **Volumes:** Mounts ./stream_config and ./recordings.
- **Command Customization:** Runs ffmpeg.sh to handle recording tasks.

4. live-overlay-server

- **Image:** restreamio/gstreamer:x86_64-latest-prod
- **Purpose:** Responsible for providing video overlay services.
- **Ports:**
- 8000:8001
- **Networking Configuration:** Connected to live_stream_network.
- **Volumes & Configuration:** Dynamically replaces environment variables in nginx.conf.template.

5. webrtc-client-viewer

- **Image:** nginx:alpine
- **Purpose:** Acts as a signaling server for WebRTC viewers.
- **Ports:** 8000:8000
- **Configuration:** Dynamically updates environment variables in nginx.conf.template.
- **Volumes:** Contains the Server directory mapped to serve as the static file viewer.

2. Networking

- **Network Name:** live_stream_network
- **Type:** bridge
- **Subnet Configuration:** Defined as 192.168.1.0/24
- This ensures all these containers can communicate over a private network without relying on public IPs.

3. Environment Configuration - .env

This file defines key server IP addresses and other configurations.

Key Variables

1. SERVER_IP_LIVE_STREAMING_SERVER_A: IP for live-streaming-server-a.
2. SERVER_IP_LIVE_STREAMING_SERVER_B: IP for live-streaming-server-b.
3. SERVER_IP_LIVE_RECORDING_SERVER: IP for the recording server.
4. SERVER_IP_LIVE_OVERLAY_SERVER: IP for the live overlay server.
5. SERVER_IP_WEBRTC_CLIENT_VIEWER: IP for WebRTC viewer signaling.
6. **RTSP Camera Configuration Variables:** Used to dynamically create camera streams.

Example values:

```
SERVER_IP_LIVE_STREAMING_SERVER_A=192.168.1.100
SERVER_IP_LIVE_STREAMING_SERVER_B=192.168.1.101
```

4. nginx.conf

This is an NGINX server configuration specific to the live overlay service (live-overlay-server) with WebRTC signaling and static file hosting.

Configuration Summary

1. **WebRTC Signaling:**

- The server listens on port 8000.
- It uses a WebSocket connection (/live/stream_key_2a) to proxy requests.

2. **Static File Serving:**

- The HTML/JS files for visualization (client side) are served from /usr/share/nginx/html.

- This is configured with root /usr/share/nginx/html; in the NGINX server block.

5. HTML Server Overlay (index.html)

This is the front-end visualization interface for live streaming.

HTML Content

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>WebRTC Live Video Overlay</title>
    <style>
      canvas {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <h1>Live Video Feed with Dynamic Timestamp Overlay</h1>
```

```
    <canvas id="videoCanvas" width="640" height="480"></canvas>
    <script src="viewer.js"></script>
  </body>
</html>
```

- **Key Component:** A canvas for drawing live video feeds.
- Includes a reference to viewer.js.

6. viewer.js

This JavaScript script manages the WebRTC peer connection, video rendering, and timestamp overlay effects.

Key Functionalities

1. **WebRTC Connection Initialization:**

- Establishes WebRTC peer connections using the signaling server URL (SIGNALING_SERVER_URL).
- Sends an offer to establish connection and listens for responses via WebSocket.

2. **Rendering Video:**

• Dynamically creates a <video> element to render video streams.

• Captures the video feed and draws it to the canvas (videoCanvas) in the front end.

3. **Dynamic Timestamp Overlay:**

• Clears and redraws the canvas every second.

• Draws the video feed and a timestamp overlay at runtime.

4. **Greenscreen Effect Placeholder:**

• Suggests future video effects could be applied (not yet implemented).

Summary of the Entire Workflow

1. **RTMP Input Streams:**

• live-streaming-server-a and live-streaming-server-b handle incoming RTMP feeds. Streams are dynamically pulled from configured camera feeds.

2. **Live Recording Server:**

• live-recording-server uses ffmpeg to record the live streams for archival purposes.

3. **WebRTC Overlay Streaming:**

• live-overlay-server overlays effects or metadata (e.g., timestamp).

4. **WebRTC Client Viewer:**

• Frontend with HTML, JS (viewer.js) connects to signaling and visualizes the video feed.

5. **Networking & Communication:**

• Docker Compose uses a private bridge network (live_stream_network) to allow services to communicate privately using their respective IPs.

Would you like a deeper dive into any specific part of the configuration, or are there any areas you'd like me to refine?