Simple Realtime Server (SRS) For RTMP Streaming

Let's dive into using Simple Realtime Server (SRS) to deploy an edge server for RTMP streaming, with a focus on understanding and configuring it effectively, including the `rtmp.conf` file. SRS is a powerful, high-performance media server designed for real-time streaming protocols like RTMP, HLS, and HTTP-FLV. It's particularly well-suited for scenarios requiring scalability, such as content delivery networks (CDNs), where an edge server can efficiently distribute streams to many clients while fetching content from an origin server. I'll walk you through the process step-by-step, explain the edge deployment concept, and clarify how to work with configurations like `rtmp.conf`.

What is SRS and Edge Deployment?

SRS is an open-source server built to handle real-time media streaming with a focus on simplicity and efficiency. It supports RTMP (Real-Time Messaging Protocol) for low-latency live streaming, alongside other protocols like HLS and WebRTC. In SRS, an "edge"server acts as an intermediary between clients (viewers or publishers) and an "origin"server (where the stream originates). The edge fetches streams from the origin only when requested by clients, reducing the load on the origin and enabling scalable distribution—think of it like a CDN's caching layer tailored for live streams.

Deploying an edge server with SRS is ideal when you need to: - Serve a large number of viewers without overwhelming the origin. - Allow publishers to push streams to the edge, which then forwards them to the origin. - Minimize bandwidth usage on costly origin servers by leveraging cheaper edge nodes.

Step-by-Step: Deploying an Edge Server with SRS for RTMP

Here's how to set up SRS as an edge server for RTMP streaming. I'll assume you're working on a Linux system (e.g., Ubuntu), as SRS is optimized for such environments.

1. Install SRS First, you'll need to get SRS running on your machine: - **Download SRS**: Grab the latest stable version from the official GitHub repository (github.com/ossrs/srs). As of today, February 26, 2025, you'd typically clone the repo: `git clone https://github.com/ossrs/srs.git   cd srs` - **Build SRS**: SRS uses a straightforward build process with `./configure` and `make`: `./configure   make` This compiles the server into the `objs` directory (e.g., `objs/srs`). - **Test the Binary**: Run it with the default config to ensure it works: `./objs/srs -c conf/srs.conf` By default, it listens on port 1935 for RTMP. Check the console output for confirmation.

2. Understand the Edge Concept In SRS, an edge server operates in "remote"mode, meaning it doesn't generate streams itself but pulls them from an origin server when a client requests them (for playback) or pushes streams to the origin (for publishing). This on-demand fetching is what makes edge servers efficient for scaling RTMP delivery.

- **Origin Server**: The source of the stream (e.g., where an encoder like OBS pushes an RTMP stream).
- **Edge Server**: A relay that clients connect to, fetching from the origin only when needed.

For this example, let's assume you have an origin server already running SRS at `192.168.1.100:1935` (replace this with your actual origin IP).

3. Configure the Edge Server SRS uses configuration files to define its behavior. The default `srs.conf` is a good starting point, but for edge deployment, you'll create a specific config—let's call it `edge.conf`. Here's how to set it up:

- **Create `edge.conf`**:

  ```
  cd conf
  nano edge.conf
  ```

- **Add Edge Configuration**: Here's a minimal `edge.conf` for RTMP edge deployment:

  ```
  listen              1935;
  max_connections     1000;
  srs_log_tank        file;
  srs_log_file        ./objs/edge.log;
  vhost __defaultVhost__ {
      cluster {
          mode        remote;
          origin      192.168.1.100:1935;
      }
  }
  ```

  - `listen 1935`: The edge listens for RTMP connections on port 1935.
  - `max_connections 1000`: Limits concurrent connections (adjust based on your server capacity).
  - `srs_log_file`: Logs to a file for debugging.
  - `vhost __defaultVhost__`: The default virtual host config.
  - `cluster { mode remote; origin 192.168.1.100:1935; }`: Sets this server as an edge (`mode remote`) and points it to the origin server.

- **Save and Exit**: Ctrl+O, Enter, Ctrl+X in nano.

4. Start the Edge Server Run SRS with your edge config:

```
./objs/srs -c conf/edge.conf
```

Check the logs (`./objs/edge.log`) to confirm it's running and connected to the origin.

5. Test the Setup

- **Publish a Stream**: Use a tool like OBS or FFmpeg to push an RTMP stream to the origin server:

  ```
  ffmpeg -re -i input.mp4 -c copy -f flv rtmp://192.168.1.100/live/livestream
  ```

  Here, `live` is the app name, and `livestream` is the stream key.

- **Play from Edge**: Use VLC or another RTMP client to play the stream from the edge:

  ```
  rtmp://<edge-server-ip>/live/livestream
  ```

  Replace `<edge-server-ip>` with your edge server's IP (e.g., `192.168.1.101`). The edge will fetch the stream from the origin and serve it to you.

6. Exploring `rtmp.conf` SRS doesn't ship with an `rtmp.conf` file by default, but you might encounter references to it in tutorials or custom setups. It's essentially a naming convention for an RTMP-specific config file. For example, the SRS documentation (ossrs.net) provides a sample `rtmp.conf` for real-time RTMP streaming:

```
listen              1935;
max_connections     1000;
vhost __defaultVhost__ {
    tcp_nodelay     on;
    min_latency     on;
    play {
        gop_cache   off;
        queue_length 10;
    }
    publish {
        mr          off;
    }
}
```

- **Purpose**: This config optimizes for low-latency RTMP streaming on an origin server, not an edge. For edge deployment, you'd adapt it by adding the `cluster` block from step 3.
- **Key Settings**:
  - `tcp_nodelay on`: Reduces latency by disabling Nagle's algorithm.
  - `min_latency on`: Prioritizes low latency over buffering.
  - `gop_cache off`: Disables caching of Group of Pictures for real-time playback.
  - `mr off`: Disables "merge read"to avoid delays in publishing.

For an edge, you'd combine this with the `cluster` settings rather than using it standalone.

Explaining More: Edge Mechanics and RTMP

- **How Edge Works**: When a client requests `rtmp://<edge-ip>/live/livestream`, the edge checks if it has the stream. If not, it pulls it from the origin (`192.168.1.100:1935`) and caches it locally to serve other clients. If a publisher pushes to the edge, it forwards the stream to the origin.
- **RTMP Specifics**: RTMP is a low-latency protocol ideal for live streaming. SRS handles RTMP efficiently, supporting features like absolute time coding (ATC) for synchronization across servers, though it's off by default in edge mode unless specified.
- **Scalability**: Add multiple edges pointing to the same origin to handle thousands of clients. SRS supports failover by listing multiple origins (e.g., `origin 192.168.1.100:1935 192.168.1.200:1935;`).

Tips and Troubleshooting

- **Firewall**: Ensure port 1935 is open on both origin and edge servers.
- **Logs**: Check `edge.log` for errors like connection failures to the origin.
- **Latency**: Edge adds minimal latency (typically <1s) if the origin is low-latency too.
- **Multiple Edges**: Deploy additional edges with the same config, adjusting `listen` ports or IPs as needed.

Wrapping Up

Deploying an SRS edge server for RTMP is straightforward once you grasp the origin-edge relationship. The `edge.conf` sets up the edge to fetch or push streams dynamically, while an `rtmp.conf`-style config could fine-tune RTMP performance if needed. With this setup, you're ready to scale live streaming efficiently—whether for a handful of viewers or a global audience. Want to tweak it further or integrate HLS alongside RTMP? Just let me know!