# Glimesh / broadcast-box Public

A broadcast, in a box.

MIT license

2.1k stars 132 forks Branches Tags Activity

Star Notifications

Code Issues 5 Pull requests 6 Actions Projects Security Insights

5 Branches 4 Tags Go to file Go to file Code ...

Sean-Der and Nabos Add Server Trickle ICE Support b0425fe · 2 days ago

| | | |
|---|---|---|
| .github | Bump actions/checkout from 5... | 2 months ago |
| examples | Add example of recording | 6 months ago |
| internal | Add Server Trickle ICE Support | 2 days ago |
| web | Bump @tailwindcss/postcss fro... | last week |
| .env.development | Revert backend rewrite | 6 months ago |
| .env.production | Revert backend rewrite | 6 months ago |
| .gitignore | Add example of recording | 6 months ago |
| CONTRIBUTING.md | Clarify readme by moving dev i... | 2 years ago |
| Dockerfile | Update Dockerfile | 3 years ago |
| LICENSE | Fix typo in LICENSE | 3 years ago |
| README.md | update webhook doc with link ... | 3 months ago |
| docker-compose.yaml | Remove ports from caddy dock... | 6 months ago |
| go.mod | Add Server Trickle ICE Support | 2 days ago |
| go.sum | Add Server Trickle ICE Support | 2 days ago |
| main.go | Add Server Trickle ICE Support | 2 days ago |

# Broadcast Box

License MIT chat 199 online

## What is Broadcast Box

Broadcast Box lets you broadcast to others in sub-second time. It was designed to be simple to use and easily modifiable. We wrote Broadcast Box to show off some of the cutting edge tech that is coming to the broadcast space.

Want to contribute to the development of Broadcast Box? See Contributing.

### Sub-second Latency

Broadcast Box uses WebRTC for broadcast and playback. By using WebRTC instead of RTMP and HLS you get the fastest experience possible.

### Latest in Video Compression

With WebRTC you get access to the latest in video codecs. With AV1 you can send the same video quality with a 50% reduction in bandwidth required.

### Broadcast all angles

WebRTC allows you to upload multiple video streams in the same session. Now you can broadcast multiple camera angles, or share interactive video experiences in real time!

### Broadcasters provide transcodes

Transcodes are necessary if you want to provide a good experience to all your users. Generating them is prohibitively expensive though, WebRTC provides a solution. With WebRTC users can upload the same video at different quality levels. This keeps things cheap for the server operator and you still can provide the same experience.

### Broadcasting for all

WebRTC means anyone can be a broadcaster. With Broadcast Box you could use broadcast software like OBS. However, another option is publishing directly from your browser! Users just getting started with streaming don't need to worry about bitrates, codecs anymore. With one press of a button you can go live right from your browser with Broadcast Box. This makes live-streaming accessible to an entirely new audience.

### Peer-to-Peer (if you need it)

With Broadcast Box you can serve your video without a public IP or forwarding ports!

Run Broadcast Box on the same machine that you are running OBS, and share your video with the world! WebRTC comes with P2P technology, so users can broadcast and playback video without paying for dedicated servers. To start the connection users will need to be able to connect to the HTTP server. After they have negotiated the session then NAT traversal begins.

You could also use P2P to pull other broadcasters into your stream. No special configuration or servers required anymore to get sub-second co-streams.

**README** Contributing MIT license

connection is established between broadcasters/viewers. If you want a direct connection between OBS and your browser see OBS2Browser.

## Using

To use Broadcast Box you don't even have to run it locally! A instance of Broadcast Box is hosted at b.siobud.com. If you wish to run it locally skip to Getting Started

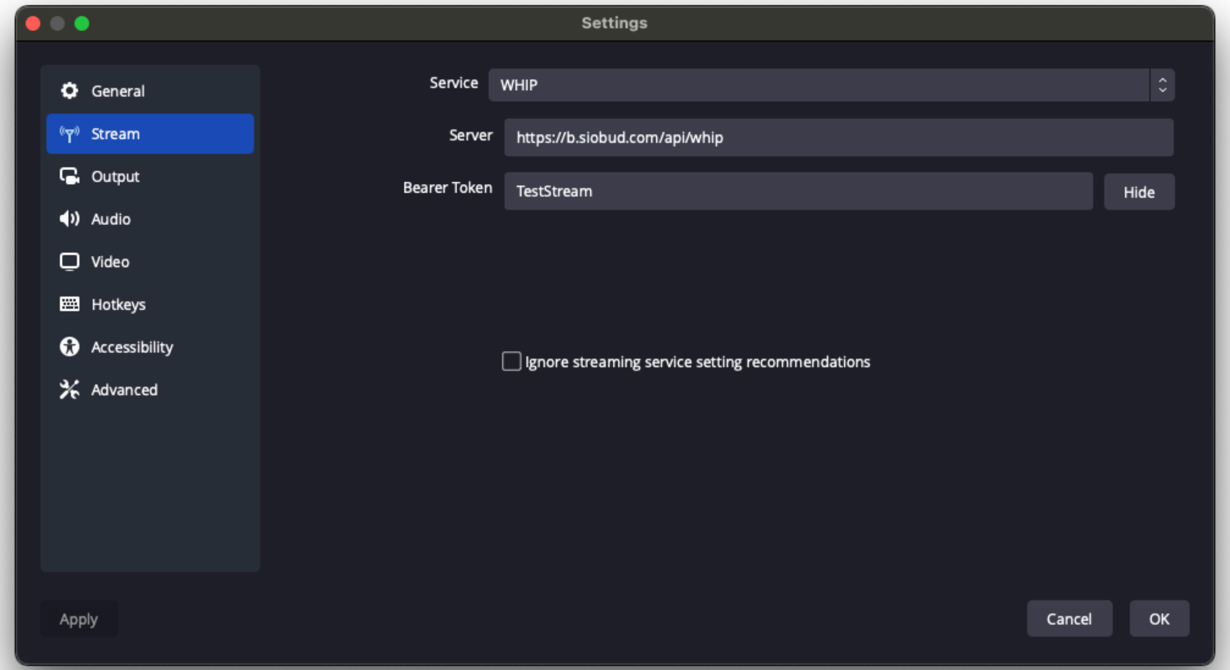
### Broadcasting

To use Broadcast Box with OBS you must set your output to WebRTC and set a proper URL + Stream Key. You may use any Stream Key you like. The same stream key is used for broadcasting and playback.

Go to `Settings -> Stream` and set the following values.

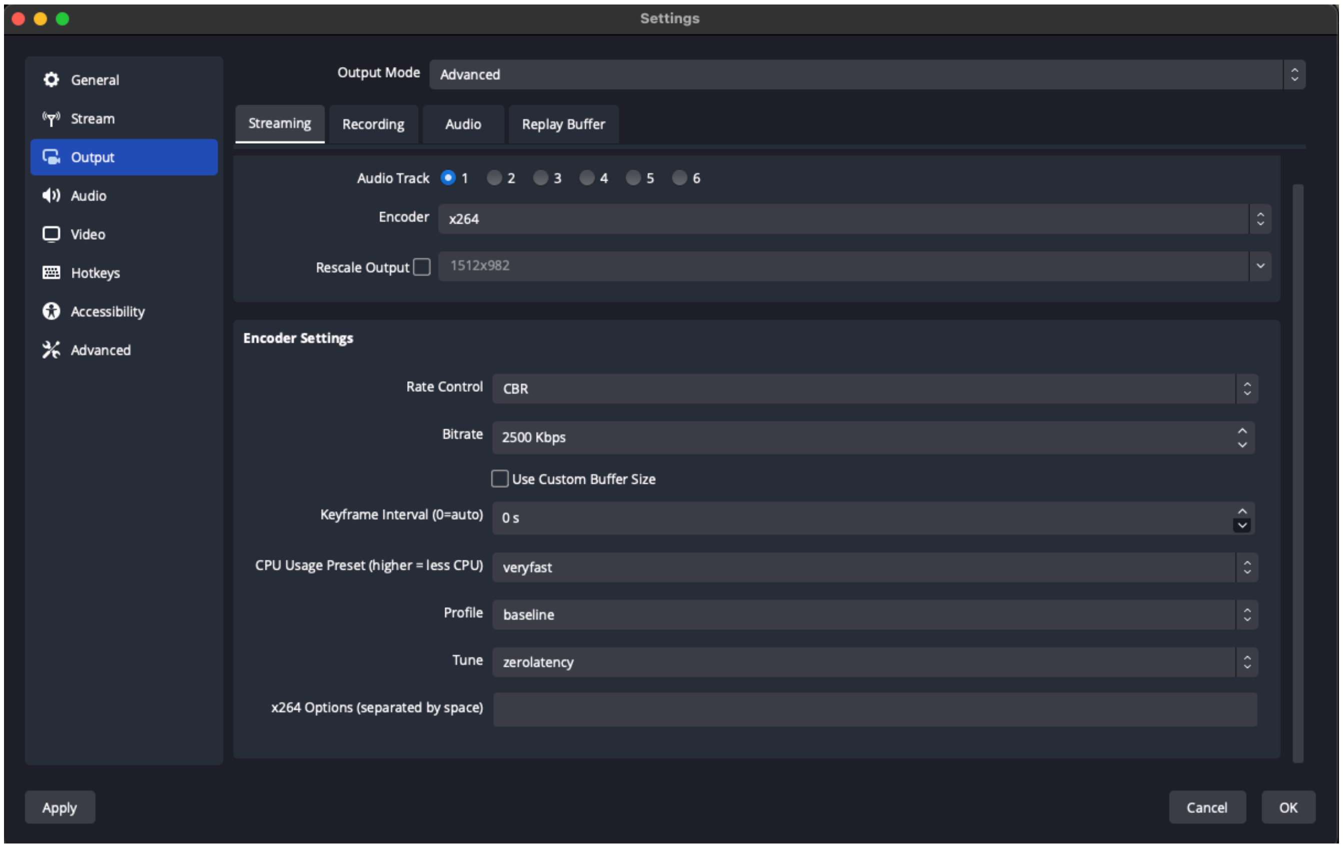
- Service: WHIP
- Server: https://b.siobud.com/api/whip
- StreamKey: (Any Stream Key you like)

Your settings page should look like this:



Service WHIP
Server https://b.siobud.com/api/whip
Bearer Token TestStream
Ignore streaming service setting recommendations

OBS by default will have ~2 seconds of latency. If you want sub-second latency you can configure this in `Settings -> Output`. Set your encoder to `x264` and set tune to `zerolatency`. Your Output page will look like this.



Output Mode: Advanced

Audio Track: [✓ 1] 2 3 4 5 6

Encoder: x264

Rescale Output ☐ 1512x982

Rate Control: CBR

Bitrate: 2500 Kbps

☐ Use Custom Buffer Size

Keyframe Interval (0=auto): 0 s

CPU Usage Preset (higher = less CPU): veryfast

Profile: baseline

Tune: zerolatency

x264 Options (separated by space):

When you are ready to broadcast press `Start Streaming` and now time to watch!

## Broadcasting (GStreamer, CLI)

See the example script here.

Can broadcast gstreamer's test sources, or pulsesrc+v4l2src

Expects `gstreamer-1.0` , with `good,bad,ugly` plugins and `gst-plugins-rs`

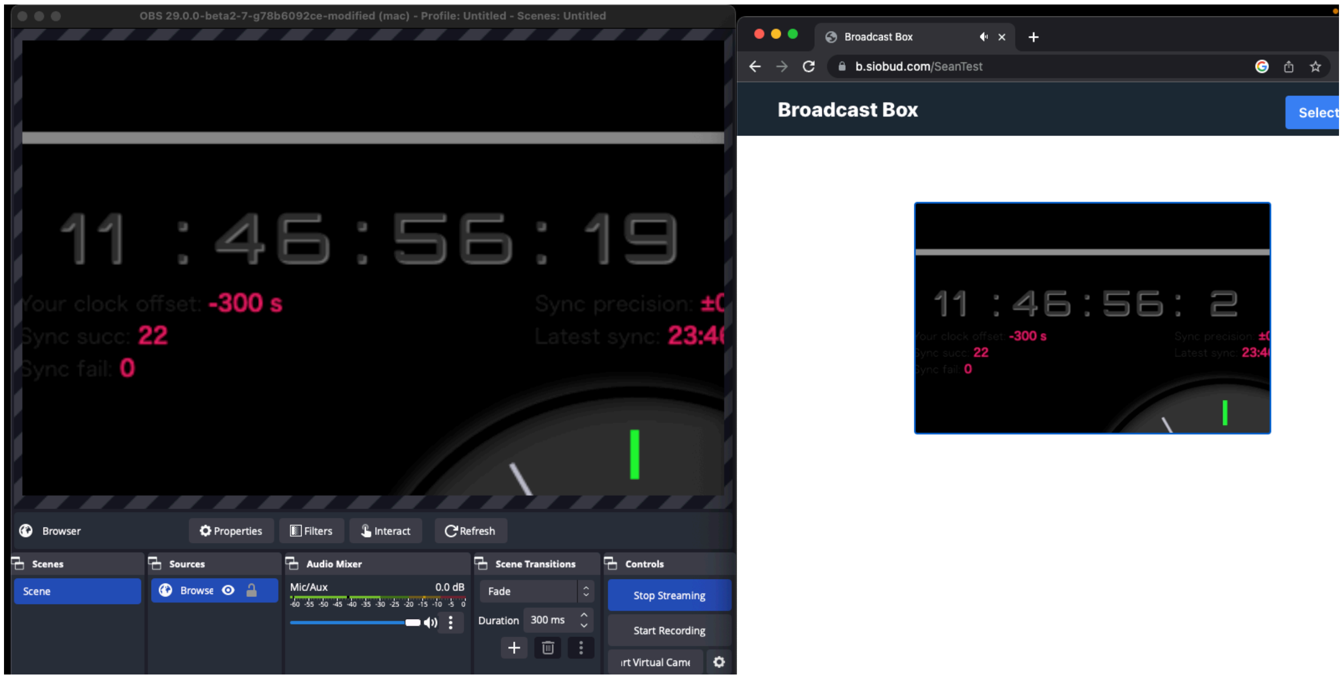
Use of example scripts:

```
# testsrcs
./examples/gstreamer-broadcast.nu http://localhost:8080/api/whip testStream1
# v4l2src
./examples/gstreamer-broadcast.nu http://localhost:8080/api/whip testStream1 v4l2
```

## Playback

If you are broadcasting to the Stream Key `StreamTest` your video will be available at https://b.siobud.com/StreamTest.

You can also go to the home page and enter `StreamTest` . The following is a screenshot of OBS broadcasting and the latency of 120 milliseconds observed.



## Getting Started

Broadcast Box is made up of two parts. The server is written in Go and is in charge of ingesting and broadcasting WebRTC. The frontend is in react and connects to the Go backend. The Go server can be used to serve the HTML/CSS/JS directly. Use the following instructions to build from source or utilize Docker / Docker Compose.

### Configuring

Configurations can be made in .env.production, although the defaults should get things going.

### Building From Source

#### Frontend

React dependencies are installed by running `npm install` in the `web` directory and `npm run build` will build the frontend.

If everything is successful, you should see the following:

```
> broadcast-box@0.1.0 build
> dotenv -e ../.env.production react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  53.51 kB  build/static/js/main.12067218.js
```

```
    2.27 kB   build/static/css/main.8738ee38.css
  ...
```

#### Backend

Go dependencies are automatically installed.

To run the Go server, run `go run .` in the root of this project, you should see the following:

```
2022/12/11 16:02:14 Loading `.env.production`
2022/12/11 16:02:14 Running HTTP Server at `:8080`
```

To use Broadcast Box navigate to: `http://<YOUR_IP>:8080`. In your broadcast tool of choice, you will broadcast to `http://<YOUR_IP>:8080/api/whip`.

### Docker

A Docker image is also provided to make it easier to run locally and in production. The arguments you run the Dockerfile with depending on if you are using it locally or a server.

If you want to run locally execute `docker run -e UDP_MUX_PORT=8080 -e NAT_1_TO_1_IP=127.0.0.1 -p 8080:8080 -p 8080:8080/udp seaduboi/broadcast-box`. This will make broadcast-box available on `http://localhost:8080`. The UDPMux is needed because Docker on macOS/Windows runs inside a NAT.

If you are running on AWS (or other cloud providers) execute. `docker run --net=host -e INCLUDE_PUBLIC_IP_IN_NAT_1_TO_1_IP=yes seaduboi/broadcast-box` broadcast-box needs to be run in net=host mode. broadcast-box listens on random UDP ports to establish sessions.

### Docker Compose

A Docker Compose is included that uses LetsEncrypt for automated HTTPS. It also includes Watchtower so your instance of Broadcast Box will be automatically updated every night. If you are running on a VPS/Cloud server this is the quickest/easiest way to get started.

```
export URL=my-server.com
docker-compose up -d
```

## URL Parameters

The frontend can be configured by passing these URL Parameters.

- `cinemaMode=true` - Forces the player into cinema mode by adding to end of URL like https://b.siobud.com/myStream?cinemaMode=true

## Environment Variables

The backend can be configured with the following environment variables.

- `WEBHOOK_URL` - URL for Webhook Backend. Provides authentication and logging
- `DISABLE_STATUS` - Disable the status API
- `DISABLE_FRONTEND` - Disable the serving of frontend. Only REST APIs + WebRTC is enabled.
- `HTTP_ADDRESS` - HTTP Server Address
- `NETWORK_TEST_ON_START` - When "true" on startup Broadcast Box will check network connectivity
- `ENABLE_HTTP_REDIRECT` - HTTP traffic will be redirect to HTTPS
- `SSL_CERT` - Path to SSL certificate if using Broadcast Box's HTTP Server
- `SSL_KEY` - Path to SSL key if using Broadcast Box's HTTP Server
- `NAT_1_TO_1_IP` - Announce IPs that don't belong to local machine (like Public IP). delineated by '|'
- `INCLUDE_PUBLIC_IP_IN_NAT_1_TO_1_IP` - Like `NAT_1_TO_1_IP` but autoconfigured
- `INTERFACE_FILTER` - Only use a certain interface for UDP traffic
- `NAT_ICE_CANDIDATE_TYPE` - By default setting a NAT_1_TO_1_IP overrides. Set this to `srflx` to instead append IPs
- `STUN_SERVERS` - List of STUN servers delineated by '|'. Useful if Broadcast Box is running behind a NAT
- `NETWORK_TYPES` - List of network types to use, delineated by '|'. Default is `udp4|udp6` .
- `INCLUDE_LOOPBACK_CANDIDATE` - Also listen for WebRTC traffic on loopback, disabled by default
- `UDP_MUX_PORT_WHEP` - Like `UDP_MUX_PORT` but only for WHEP traffic
- `UDP_MUX_PORT_WHIP` - Like `UDP_MUX_PORT` but only for WHIP traffic
- `UDP_MUX_PORT` - Serve all UDP traffic via one port. By default Broadcast Box listens on a random port
- `TCP_MUX_ADDRESS` - If you wish to make WebRTC traffic available via TCP.
- `TCP_MUX_FORCE` - If you wish to make WebRTC traffic only available via TCP.

- `APPEND_CANDIDATE` - Append candidates to Offer that ICE Agent did not generate. Worse version of `NAT_1_TO_1_IP`
- `DEBUG_PRINT_OFFER` - Print WebRTC Offers from client to Broadcast Box. Debug things like accepted codecs.
- `DEBUG_PRINT_ANSWER` - Print WebRTC Answers from Broadcast Box to Browser. Debug things like IP/Ports returned to client.

## Authentication and Logging

To prevent random users from streaming to your server, you can set the `WEBHOOK_URL` and validate/process requests in your code. This enables you to separate the authorization between broadcasting (whip) and watching (whep). So you can safely share a watch link without exposing the key used for broadcasting.

If the request succeeds (meaning the stream key is accepted), broadcast-box redirects the stream to an url given by the external server, otherwise the streaming request is dropped.

See here. For an example Webhook Server that only allows the stream `broadcastBoxRulez`

For a more advanced example of a webhook server implementation making use of separating the key for streaming from the key for watching, see the broadcastbox-webhookserver repository.

## Network Test on Start

When running in Docker Broadcast Box runs a network tests on startup. This tests that WebRTC traffic can be established against your server. If you server is misconfigured Broadcast Box will not start.

If the network test is enabled this will be printed on startup

```
NETWORK_TEST_ON_START is enabled. If the test fails Broadcast Box will exit.
See the README.md for how to debug or disable NETWORK_TEST_ON_START
```

If the test passed you will see

```
Network Test passed.
Have fun using Broadcast Box
```

**Releases**

**4** tags

## Packages

No packages published

## Contributors 24

+ 10 contributors

## Languages

● **Go** 50.0% ● **TypeScript** 47.4% ● **Other** 2.6%