ChourangiHealth Video Consultation Platform

Full Project Documentation

From Concept to Production Deployment

Version 1.0 | February 2026

Production URL: https://chourangi.duckdns.org

1. Project Overview

This document covers the complete implementation of the ChourangiHealth Video Consultation Platform, a real-time doctor-patient video calling system built using OpenVidu, FastAPI, and React. The project went from initial concept to a fully working production deployment on AWS EC2 with features including video calling, real-time chat, and session recording.

1.1 What Was Built

- Secure doctor-patient video consultation system
- Real-time bidirectional chat during video calls
- Session recording with direct MP4 download
- JWT-based authentication with role-based access (Doctor and Patient)
- Production deployment on AWS EC2 with HTTPS and a custom domain
- TURN server configuration for reliable WebRTC connectivity

1.2 Technology Stack

- Frontend: React.js with OpenVidu Browser SDK
- Backend: Python FastAPI
- Video Engine: OpenVidu v2.32 (self-hosted on EC2)
- Media Server: Kurento Media Server 7.3.0
- TURN Server: COTURN (built into OpenVidu)
- Infrastructure: AWS EC2 c6a.xlarge, Ubuntu 24
- Domain: chourangi.duckdns.org (DuckDNS free domain)
- SSL: Let's Encrypt via Certbot
- Reverse Proxy: Nginx (built into OpenVidu)

2. Infrastructure and Server Setup

2.1 AWS EC2 Instance

- Instance Type: c6a.xlarge (4 vCPUs, 8 GB RAM)
- Operating System: Ubuntu 24.04 LTS
- Region: eu-west-2 (London)
- Public IP: 18.130.149.212
- Key pair authentication for SSH access

2.2 Security Group Configuration

The following inbound rules were configured on the AWS Security Group to allow OpenVidu and WebRTC traffic:

- Port 22 TCP: SSH access (IPv4 and IPv6)
- Port 80 TCP: HTTP (IPv4 and IPv6)
- Port 443 TCP: HTTPS (IPv4 and IPv6)
- Port 3478 UDP/TCP: TURN/STUN server (IPv4 and IPv6)
- Port 40000-65535 UDP/TCP: WebRTC media ports and KMS RTP ports (IPv4 and IPv6)
- Port 4443 TCP: OpenVidu server (initially, later changed to 5443 via Nginx)

2.3 Domain and SSL Setup

A free dynamic DNS domain was registered at DuckDNS (duckdns.org). The domain chourangi.duckdns.org was pointed to the EC2 public IP 18.130.149.212. Let's Encrypt SSL certificate was automatically provisioned by OpenVidu during initial setup using the domain and a valid email address.

2.4 OpenVidu Installation

OpenVidu v2.32 was installed on the EC2 instance using the official installation script. The installation placed all configuration files in /opt/openvidu/. The main configuration file is /opt/openvidu/.env.

Key configuration values set in /opt/openvidu/.env:

```
DOMAIN_OR_PUBLIC_IP=chourangi.duckdns.org
CERTIFICATE_TYPE=letsencrypt
LETSENCRYPT_EMAIL=ashutosh@xtensible.in
OPENVIDU_SECRET=openvidu-v2-32
COTURN_IP=18.130.149.212
KMS_PUBLIC_IP=18.130.149.212
OPENVIDU_RECORDING=true
OPENVIDU_RECORDING_PATH=/opt/openvidu/recordings
```

OpenVidu runs as a set of Docker containers managed via docker-compose. The containers are: openvidu-server, kms (Kurento), coturn, nginx, and app.

3. Critical WebRTC and ICE Configuration Fix

This was the most important and time-consuming part of the project. After initial deployment, video calls were failing with ICE_CONNECTION_DISCONNECTED errors. This section documents the root cause and the fix applied.

3.1 The Problem

Kurento Media Server (KMS) runs inside Docker on EC2. It only knew its private/internal IP address (172.31.33.5) and was advertising this private IP to browsers in its ICE candidates. Browsers on the internet cannot reach a private IP, so WebRTC connections were failing with ICE state FAILED after a few seconds.

3.2 Debugging Steps Taken

- Checked COTURN logs which showed remote 18.130.149.212 as client IP meaning traffic was looping back through the EC2 public IP
- Verified COTURN was in bridge network mode causing wrong IP mapping
- Changed COTURN to host network mode so it could see real client IPs
- Added --external-ip=18.130.149.212/172.31.33.5 to COTURN command
- Added --listening-ip=0.0.0.0 so COTURN accepts all connections
- Discovered KMS was using eth0 interface name which is wrong on AWS EC2 (AWS uses ens5)
- Found KMS config file at /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini inside the container
- Found that KMS container reads settings from environment variables not config file directly

3.3 The Final Fix

The fix required adding three environment variables to the KMS container in /opt/openvidu/docker-compose.yml under the kms service:

```
KMS_EXTERNAL_IPV4=18.130.149.212
KMS_NETWORK_INTERFACES=ens5
KMS_ICE_TCP=0
```

KMS_EXTERNAL_IPV4 tells KMS to advertise the public IP in its ICE candidates instead of the private IP. KMS_NETWORK_INTERFACES=ens5 tells KMS to use the correct network interface on AWS EC2 (which is ens5, not eth0). KMS_ICE_TCP=0 disables TCP ICE candidates which were causing additional issues.

Once these settings were applied and the KMS container was force-recreated, video calls started working correctly between devices on different networks.

3.4 Backend URL Fix

The backend was also misconfigured to call OpenVidu through the public domain which caused timing issues. The fix was to change OPENVIDU_URL in the backend .env file to call OpenVidu directly on localhost:

```
OPENVIDU_URL=http://localhost:5443
```

OpenVidu server listens on port 5443 (HTTP) internally. Nginx proxies external HTTPS traffic to this port.

4. Backend Implementation (FastAPI)

4.1 Project Structure

```
backend/
  app/
    main.py          - FastAPI app entry point
    config.py        - Environment variable configuration
    dependencies.py  - JWT token verification
    routers/
      auth.py        - Login endpoint
      video.py       - Video session and recording endpoints
    services/
      auth_service.py     - Authentication logic
      openvidu_service.py - OpenVidu API calls
    schemas/
      auth_schema.py  - Login request/response models
      video_schema.py - Video session request/response models
  .env               - Environment variables
```

4.2 Environment Variables

```
OPENVIDU_URL=http://localhost:5443
OPENVIDU_USERNAME=OPENVIDUAPP
OPENVIDU_SECRET=openvidu-v2-32
JWT_SECRET=supersecretjwtkey
JWT_ALGORITHM=HS256
```

4.3 API Endpoints

- POST /api/auth/login: Takes username and password, returns JWT token with role claim
- POST /api/video/session: Creates or gets an OpenVidu session and returns a connection token
- POST /api/video/recording/start: Starts COMPOSED recording for a session
- POST /api/video/recording/stop: Stops recording and returns the MP4 URL
- GET /api/video/recording/{recording_id}: Gets recording status and URL
- GET /api/video/recording/{recording_id}/download: Proxies the MP4 download through the backend without requiring OpenVidu credentials
- GET /health: Health check endpoint

4.4 Key Implementation Details

Session Creation with 409 Handling: When both doctor and patient join the same calling ID, the second request gets a 409 Conflict from OpenVidu because the session already exists. The openvidu_service.py handles this correctly by returning the session ID directly when a 409 is received instead of raising an error.

Recording Download Proxy: The recording MP4 files are stored on the OpenVidu server and require OpenVidu admin credentials to access directly. To avoid exposing credentials to the browser or showing a login popup, a download endpoint was created in the backend that fetches the file from OpenVidu using httpx with the stored credentials and streams it back to the browser with a Content-Disposition attachment header triggering a direct download.

4.5 Running the Backend on EC2

```
cd /home/ubuntu/test-1/backend
nohup uvicorn app.main:app --host 0.0.0.0 --port 8000 > /tmp/backend.log 2>&1 &
```

The backend runs on port 8000. Nginx proxies /api/ requests from the public domain to http://172.17.0.1:8000 (Docker gateway IP to reach the host).

5. Frontend Implementation (React)

5.1 Project Structure

```
frontend/
  src/
    App.js          - Root component with auth routing
    index.js        - React entry point
    index.css       - Global styles (overflow hidden for fixed layout)
    api/
      api.js        - All API calls to backend
    pages/
      LoginPage.js   - Login screen
      VideoPage.js   - Main video consultation page
```

5.2 Key Features in VideoPage.js

- Join screen with role-specific avatars for Doctor and Patient
- Real-time video using OpenVidu Browser SDK with publisher and subscriber streams
- Picture-in-picture layout with remote video fullscreen and local video in corner
- Call timer that starts when both participants are connected
- Video on/off and audio mute/unmute controls
- Real-time WhatsApp-style chat panel using OpenVidu Signal API
- Unread message badge on Chat button when chat panel is closed
- Recording controls: Start Recording, Stop Recording, and Download buttons
- Leave session button with full cleanup

5.3 Chat Implementation

Chat was implemented using the OpenVidu Signal API which sends messages through the existing WebSocket connection. Each message is sent as a signal of type chat with a JSON payload containing the message text, sender name, role, and timestamp. A listener on the session picks up incoming signals and adds them to the messages state. Messages from the sender themselves are excluded from the signal listener since the sender already adds their own message to state immediately on send.

A known issue was fixed where using messagesEndRef.current.scrollIntoView was causing the entire page to scroll. This was replaced with a chatMessagesRef on the messages container that sets scrollTop to scrollHeight directly, causing only the chat panel to scroll.

5.4 Recording UI Flow

When a session is active, a Record button appears in the header for both Doctor and Patient. Clicking Record calls the backend start recording endpoint and changes the button to Stop Rec with a pulsing animation indicating recording is in progress. Clicking Stop Rec calls the stop recording endpoint and changes the button to Download. Clicking Download calls the backend download endpoint which proxies the MP4 file directly to the browser as a file download. After download completes the button resets back to Record so the cycle can repeat.

5.5 Layout and Styling

The layout uses position fixed with height 100vh and overflow hidden to prevent any page scrolling. The header is fixed at the top. The main area uses flexbox row layout with the video section and chat panel side by side. The chat panel slides in from the right when opened without disrupting the video layout. All CSS is written as inline JavaScript style objects using a styles constant at the bottom of the file.

5.6 Important Fix for React StrictMode

React StrictMode in development causes useEffect to run twice which was duplicating event listeners. The fix was to use a ref (sessionRef) to track the session object and check it before adding listeners, and to always clean up on unmount. The isChatOpenRef was added as a ref mirror of the isChatOpen state so the signal event listener closure could always read the current value without going stale.

6. Deployment Process

6.1 Repository Structure

- Frontend GitHub: https://github.com/AshuuuPatil/openvidu_videocall_microservice_frontend
- Backend GitHub: https://github.com/AshuuuPatil/openvidu_videocall_microservice_backend

6.2 Frontend Deployment Steps

The frontend is built as a static React app and served by the OpenVidu Nginx container from the custom-layout directory.

```
cd /home/ubuntu/test-1/frontend
git fetch origin master && git reset --hard origin/master
npm run build
cp -r build/* /opt/openvidu/custom-layout/
```

6.3 Backend Deployment Steps

```
cd /home/ubuntu/test-1/backend
git fetch origin master && git reset --hard origin/master
pkill -f uvicorn
nohup uvicorn app.main:app --host 0.0.0.0 --port 8000 > /tmp/backend.log 2>&1 &
```

6.4 Nginx Configuration

A custom Nginx location file at /opt/openvidu/custom-nginx-locations/frontend.conf handles routing:

- / serves the React build from /opt/openvidu/custom-layout/
- /api/ proxies to http://172.17.0.1:8000 (the FastAPI backend)

172.17.0.1 is the Docker bridge gateway IP which routes to the host machine where uvicorn is running.

6.5 OpenVidu Management Commands

```
cd /opt/openvidu
./openvidu start    # Start all containers
./openvidu stop     # Stop all containers
./openvidu restart  # Restart all containers
./openvidu logs     # View OpenVidu server logs
```

7. Recording Feature

7.1 How Recording Works

OpenVidu v2.32 has built-in recording support powered by Kurento. COMPOSED mode records all publisher streams in a single MP4 file in a grid layout similar to Zoom. The grid automatically adjusts based on the number of active publishers using BEST_FIT layout. If 2 people are publishing video, the recording shows them side by side. If 3 people join, it shows a 3-panel grid.

7.2 Recording Configuration

Recording was enabled by adding these lines to /opt/openvidu/.env:

```
OPENVIDU_RECORDING=true
OPENVIDU_RECORDING_PATH=/opt/openvidu/recordings
```

After changing the .env, OpenVidu was restarted for the changes to take effect.

7.3 Recording API

Start Recording: POST to /openvidu/api/recordings/start with body containing session ID, outputMode COMPOSED, hasAudio true, hasVideo true, and recordingLayout BEST_FIT. Returns a recording object with an ID.

Stop Recording: POST to /openvidu/api/recordings/stop/{recordingId}. Returns the recording object with status ready and a URL pointing to the MP4 file.

The MP4 files are stored at /opt/openvidu/recordings/ on the EC2 server and are accessible via the OpenVidu Nginx at https://chourangi.duckdns.org/openvidu/recordings/{id}/{id}.mp4 but require OpenVidu admin credentials.

7.4 Download Without Credentials

To provide a seamless download experience, a proxy endpoint was created in the FastAPI backend at GET /api/video/recording/{recording_id}/download. This endpoint uses the httpx library to fetch the MP4 from OpenVidu internally using stored admin credentials and streams the content back to the browser with Content-Disposition: attachment header. This causes the browser to download the file directly without ever showing a credentials popup. The httpx library was installed on EC2 with: pip install httpx --break-system-packages

8. Issues Encountered and Resolutions

8.1 Video Not Working Between Different Devices

Root Cause: KMS was advertising its private Docker IP in ICE candidates instead of the EC2 public IP. Resolution: Added KMS_EXTERNAL_IPV4=18.130.149.212 environment variable to the KMS container. Also corrected the network interface from eth0 to ens5 which is the correct interface name on AWS EC2.

8.2 COTURN Not Relaying Media

Root Cause: COTURN container was in bridge network mode so it could not see real client IPs. All connections appeared to come from the EC2 public IP itself. Resolution: Added network_mode: host to the COTURN container in docker-compose.yml and added --listening-ip=0.0.0.0 and --external-ip=18.130.149.212/172.31.33.5 flags.

8.3 Backend 500 Error on Session Creation

Root Cause: The openvidu_service.py had two create_session methods defined. Python kept only the last one. Additionally the 409 response was being handled incorrectly returning a dict without the id key that the connection endpoint expected. Resolution: Removed the duplicate method and fixed the 409 handler to return the correct dict format.

8.4 Git Conflicts with Python Cache Files

Root Cause: Python .pyc bytecode files and __pycache__ directories were being tracked by git causing merge conflicts on every pull. Resolution: Added **/__pycache__/ and **/*.pyc to .gitignore and removed the cached files from git tracking using git rm -r --cached.

8.5 Chat Messages Causing Page Scroll

Root Cause: Using messagesEndRef.current.scrollIntoView() was scrolling the entire page not just chat panel. Resolution: Replaced with a ref on the chat messages container div and used chatMessagesRef.current.scrollTop = chatMessagesRef.current.scrollHeight to scroll chat panel.

8.6 Recording Download Showing Credentials Popup

Root Cause: The frontend was redirecting directly to the OpenVidu MP4 URL which requires HTTP Basic authentication. Resolution: Created a proxy download endpoint in the backend that fetches the file with stored credentials and streams it to the browser with an attachment header.

8.7 Frontend GitHub Pull Requiring Password Every Time

The frontend GitHub repository required password authentication on EC2 for every pull. This is because the repository was cloned with HTTPS. For production deployment this was handled manually each time. For a permanent fix SSH key authentication should be configured.

9. Important File Paths Reference

9.1 EC2 Server Paths

- OpenVidu configuration: /opt/openvidu/.env
- OpenVidu docker-compose: /opt/openvidu/docker-compose.yml
- OpenVidu management script: /opt/openvidu/openvidu
- Nginx custom locations: /opt/openvidu/custom-nginx-locations/frontend.conf
- Frontend build served from: /opt/openvidu/custom-layout/
- Recordings stored at: /opt/openvidu/recordings/
- Backend code: /home/ubuntu/test-1/backend/
- Frontend code: /home/ubuntu/test-1/frontend/
- Backend logs: /tmp/backend.log
- OpenVidu startup logs: /tmp/openvidu.log

9.2 Local Development Paths

- Backend: D:\call.chourangihealth\2) POC\1)Code\openvidu-fastapi-poc\
- Frontend: D:\call.chourangihealth\2) POC\1)Code\openvidu-frontend-poc\

9.3 Production URLs

- Main application: https://chourangi.duckdns.org
- OpenVidu Dashboard: https://chourangi.duckdns.org/dashboard
- Backend health check: https://chourangi.duckdns.org/api/health (via Nginx proxy)
- Backend Swagger docs (local): http://localhost:8000/docs
- Recordings: https://chourangi.duckdns.org/openvidu/recordings/{id}/{id}.mp4

10. Credentials and Access

10.1 Application Login

- Doctor login: username doctor, password doctor123, role admin
- Patient login: username patient, password patient123, role user

10.2 OpenVidu Admin

- Username: OPENVIDUAPP
- Password: openvidu-v2-32
- Dashboard: https://chourangi.duckdns.org/dashboard

10.3 EC2 SSH Access

- User: ubuntu
- IP: 18.130.149.212
- Authentication: Key pair (PEM file)

10.4 DuckDNS

- Domain: chourangi.duckdns.org
- Points to: 18.130.149.212

11. Complete Project Summary

This project successfully delivered a fully functional real-time video consultation platform for ChourangiHealth. The platform allows doctors and patients to join a shared video room using a calling ID, communicate via video and audio, send real-time chat messages, and record sessions for later review.

The most significant technical challenge was configuring WebRTC connectivity on AWS EC2. AWS EC2 instances do not have the public IP directly on their network interface and instead use NAT. This meant that KMS was advertising unreachable private IPs to browsers. The solution required deep debugging of Kurento and COTURN configurations, including setting the correct external IP, the correct network interface name specific to AWS (ens5 instead of eth0), and switching COTURN to host network mode.

The chat feature was built using the OpenVidu Signal API which allows custom messages to be sent over the existing WebSocket connection without any additional infrastructure. The recording feature leverages OpenVidu's built-in COMPOSED recording which produces a single MP4 file containing all participants' video and audio streams in a grid layout.

All code is version controlled in GitHub, deployed to AWS EC2, and served over HTTPS with a Let's Encrypt certificate. The system is stable and ready for further feature development.