Openvidu 배포 및 실행

Openvidu 서버용 index.js

```
require("dotenv").config(!!process.env.CONFIG ? {path: process.env.CONFIG} : {});
var express = require("express");
var bodyParser = require("body-parser");
var https = require("https");
var OpenVidu = require("openvidu-node-client").OpenVidu;
var socketIo = require('socket.io');
var cors = require("cors");
var axios = require('axios');
var base64 = require('base64-js');
var app = express();
const fs = require("fs");


// 노드서버
var SERVER_PORT = process.env.SERVER_PORT;
// openvidu 서버
var OPENVIDU_URL = process.env.OPENVIDU_URL;
// opvidu 서버 비번
var OPENVIDU_SECRET = process.env.OPENVIDU_SECRET;

const privateKey = fs.readFileSync("privkey.pem", "utf8");
const certificate = fs.readFileSync("cert.pem", "utf8")
const ca = fs.readFileSync("fullchain.pem", "utf8")

const credentials = {
    key: privateKey,
    cert: certificate,
    ca: ca
};

// Cors 지금은 전체인데 나중에 바꿔줘야함
app.use(
  cors({
    // origin: "*",
        origin: "https://i9b206.p.ssafy.io",
  })
);


// express 로 HTTP 서버를 생성
var server = https.createServer(credentials, app);
// openvidu 클라이언트 객체를 초기화
var openvidu = new OpenVidu(OPENVIDU_URL, OPENVIDU_SECRET);
openvidu.activeSessions = [];
// socket.io 초기화
// var io = socketIo(server);
var io = socketIo(server, {
  cors: {
    origin: "https://i9b206.p.ssafy.io",
    methods: ["GET", "POST", "PUT", "DELETE"],
    credentials: true,
  },
});

// express 앱에 bodyParser 미들웨어를 추가하여 POST 요청의 body를 쉽게 파싱
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

// '/public' 디렉토리의 정적 파일을 호스팅
app.use(express.static(__dirname + '/public'));
```

```
// 지정된 포트에서 HTTP 서버를 시작
server.listen(SERVER_PORT, () => {
  console.log("Application started on port: ", SERVER_PORT);
  console.warn('Application server connecting to OpenVidu at ' + OPENVIDU_URL);
});

// openvidu 세션을 생성하는 API 엔드포인트를 정의
app.post("/api/sessions", async (req, res) => {
  try {
      const session = await openvidu.createSession();
      res.send(session.sessionId);
  } catch (error) {
      res.status(500).send("Error creating session: " + error.message);
  }
});

// 지정된 세션 ID에 연결을 생성하는 API 엔드포인트를 정의
app.post("/api/sessions/:sessionId/connections", async (req, res) => {
  console.log("0번");
  console.log("Request params:", req.params);
  console.log("Request body:", req.body);

  const sessionId = req.params.sessionId;

  let session;
  try {
      session = await openvidu.createSession({ customSessionId: sessionId });
  } catch (error) {
      console.error("Error fetching the session:", error);
      res.status(500).send("Error fetching the session: " + error.message);
      return;
  }

  try {
      const token = await session.generateToken();
      console.log("Token created:", token);
      res.send(token);
  } catch (error) {
      console.error("Error creating token:", error);
      res.status(500).send("Error creating token: " + error.message);
  }
});

// 세션의 스트림 정보를 가져오는 함수
async function getSessionInfo(sessionId) {
    const authString = "Basic " + base64.fromByteArray(Buffer.from("OPENVIDUAPP:" + OPENVIDU_SECRET));
    // console.log("Fetching session info for sessionId:", sessionId);  // 로그 추가
    try {
        const response = await axios.get(OPENVIDU_URL + "/api/sessions/" + sessionId, {
            headers: {
                Authorization: authString
            }
        });
        // console.log("Got response:", response.data);  // 로그 추가
        // console.log("여기",response.data.connections.content);
        return response.data.connections.content;
    } catch (error) {
        console.error("Error fetching session info:", error);
        return null;
    }
}

// API 엔드포인트 추가: 세션의 스트림 정보를 가져오기
app.get("/api/sessions/:sessionId/streams", async (req, res) => {
    try {
        const sessionId = req.params.sessionId;
        const connections = await getSessionInfo(sessionId);

        if (connections && connections.length > 0) {
            const streamIds = connections.flatMap(connection =>
                connection.publishers.map(p => p.streamId)
            );
```

```
            res.json(streamIds);
        } else {
            res.status(500).send("Error fetching streams or no active streams");
        }
    } catch (error) {
        console.error("Endpoint error:", error);
        res.status(500).send("Unexpected error");
    }
});



// 미처리된 예외를 처리
process.on('uncaughtException', err => console.error(err));

// 유저 세션 초기화
var userSessions = {};



// 여긴 p2p 시그널링서버 이벤트 핸들러

io.on('connection', function(socket) {
    socket.on('offer', function(data){
        const targetSocketId = Object.keys(userSessions).find(key => userSessions[key].userId === data.target);
        if (targetSocketId) {
            socket.to(targetSocketId).emit('offer', data);
        }
        console.log("22222222222222222222222222222222222")
        console.log(data.target,"offer")
    });

    socket.on('answer', function(data) {
        const targetSocketId = Object.keys(userSessions).find(key => userSessions[key].userId === data.target);
        if (targetSocketId) {
            socket.to(targetSocketId).emit('answer', data);
        }
        console.log("33333333333333333333333333333333333333333");
        console.log(data.target,"answer")
    });

    socket.on('ice-candidate', function(data) {
        const targetSocketId = Object.keys(userSessions).find(key => userSessions[key].userId === data.target);
        if (targetSocketId) {
            socket.to(targetSocketId).emit('ice-candidate', data);
        }
        console.log("44444444444444444444444444444444444444");
        console.log(data.target, "ice");
    });

    socket.on('join-session', function(data) {
        // 1. 현재 세션에 참여하고 있는 다른 사용자들의 ID를 가져옵니다.
        const otherUserIdsInSession = Object.values(userSessions)
        .filter(session => session.sessionId === data.sessionId && session.userId !== data.userId)
        .map(session => session.userId);

        // 2. 새로운 사용자에게 현재 세션에 참여하고 있는 다른 모든 사용자들의 ID를 전송합니다.
        socket.emit('current-users', { userIds: otherUserIdsInSession });

        // 3. 다른 사용자들에게 새로운 사용자가 참여했다는 것을 알립니다.
        socket.to(data.sessionId).emit('new-peer', { userId: data.userId });

        // 유저의 세션 정보를 저장
        userSessions[socket.id] = {
        userId: data.userId,
        sessionId: data.sessionId
        };

        console.log(userSessions[socket.id], "@@@@@@@@@@@@@@@@@@@@@@"); // 현재 소켓의 세션 정보 출력
        console.log(userSessions, "###### Entire userSessions object ######"); // 전체 userSessions 객체 출력

        // socket.io의 방을 사용하여 세션별로 유저들을 그룹화
```

```
        socket.join(data.sessionId);
    });

    socket.on('disconnect-user', function() {
        console.log('유저 접속해제:', socket.id);

        // 유저의 세션 참가 정보를 제거
        if (userSessions[socket.id]) {
            const userSession = userSessions[socket.id];
            socket.leave(userSession.sessionId);

            console.log('유저 :', userSession.userId, '세션 :', userSession.sessionId);

            delete userSessions[socket.id];
        }
    });``
});
```

Dockerfile

```
FROM node:18.16.1 as build
# FROM node:16-alpine3.16

WORKDIR /opt/openvidu-basic-node

# Copy openvidu-basic-node
COPY . /opt/openvidu-basic-node

ENV REACT_APP_REST_API_KEY=445d7747c20f75d77416c3c4ea2dca4e
ENV REACT_APP_REDIRECT_URL=https://i9b206.p.ssafy.io/kakao
ENV REACT_APP_BACKEND_SERVER_URL=https://i9b206.p.ssafy.io:9090
ENV OPENVIDU_URL=https://i9b206.p.ssafy.io:8445
ENV REACT_APP_FRONT_SERVER=https://i9b206.p.ssafy.io:5000
ENV OPENVIDU_SECRET=PWB206206
ENV SERVER_PORT=5000

# Install openvidu-basic-node dependencies
RUN npm --prefix /opt/openvidu-basic-node install
RUN npm install axios@1.4.0

ENTRYPOINT [ "/usr/local/bin/node", "index.js" ]
```

도커 이미지 제작

```
sudo docker build . -t openvidu-basic-node
```

컨테이너 실행

```
docker run -d -p 5000:5000 --name openvidu-basic-node openvidu-basic-node
```

SSL 적용

💡 ***같은 위치에 privkey.pem, cert.pem, pullchain.pem 있어야함!***