WEBRTC

태그	SKILL
담당자	지현 김 JUBI

RREFERENCE

RREFERENCE

- 1:1 실시간 채팅 서버 설계
  - 클라이언트 연결 관점

> 채팅 서버 설계를 위한 배경지식 정리 (HTTP, WebSocket, WebRTC)
> ===== 채팅서버 구현하기 시리즈 ===== 👉 [Web-Network] - 채팅 서버 설계를 위한 배경지식 정리 (HTTP, WebSocket, WebRTC) ✔️ [Spring/Spring Boot] - Spring WebSocket 공식문서 가이드 살펴보기 ✔️ [Spring/Spring
> https://thalals.tistory.com/442

- 구현 방식 종류

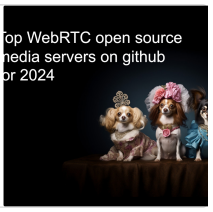
> [WebRTC] WebRTC의 다양한 방식들 (Mesh, SFU, MCU)
> P2P(Mesh) 이전 포스트에서 설명했던 P2P 방식이다. 클라이언트 즉, Peer 간의 연결을 진행하기 때문에 서버는 단순 연결을 위한 정보를 중계할 때만 사용된다. 위의 그림에서처럼 클라이언트마다 자신
> https://gilssang97.tistory.com/80

- Open source 추천 : 미디어 서버

Top WebRTC open source media servers on github for 2024

What are the WebRTC open source media servers in 2024, and which ones are the best, based on github stars.

https://bloggeek.me/webrtc-open-source-media-servers-github-2024/

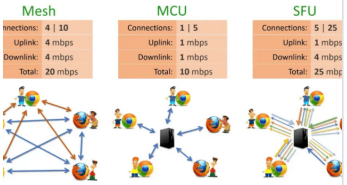


- WebRTC :: Signaling Server

[언택트 기술 시리즈]webrtc 서버 구축 1편 -기초

아자르, 스무디, 행아웃 언택트 시대하면 위 서비스들을 한번쯤은 들어보거나 사용해봤을 것이다. 맞다. 위 서비스들은 webrtc로 만들어진 서비스 들이다. 'webrtc = 영

https://kid-dev.tistory.com/4



1️⃣ WebRTC 구현 방식

① 종류

> Mesh, SFU, MCU

	Mesh (P2P)	SFU	MCU
장점	- 서버 부하가 적다 - 실시간 송수신 보장	- 클라이언트 부하 감소 - 실시간 송수신 보장 (Mesh보단 떨어짐)	- 클라이언트 부하 많이 감소 - N:M 연결에 효율적
단점	- N:M 연결 확장 시, 클라이언트 부하가 심해짐	- 서버 부하 증가 - 클라이언트 부하 여전히 높음	- 실시간 송수신 보장 ❌ - 서버 부하 많이 증가
	시그널링 서버	미디어 서버	미디어 서버

② FODONG 에서의 구현 방식 : SFU

FODONG 의 환경

- 실시간 데이터 송수신이 필요한 상황 : 여러 참여자가 함께 동화구연 함
- 실시간 데이터 송수신이 필요 없는 상황 : 한 명만 동화구연 함
- 실시간이 보장되는 환경에서 여러 명이 접속해서 비디오 및 음성 데이터들을 주고 받아야 한다
- 참여 인원은 적을 수도 많을 수도 있다
- 한 번 방송에 몇십만 명이 시청하는 서비스보다는

  클럽하우스나 아프리카TV처럼 다수의 방송이 독립적으로 존재하고

  2-4명이 대화하는 서비스가 주를 이룬다

  → 개별 방에 대한 트랜잭션을 미리 예측하거나 대응하기보다는

  → 확장성(Scalability)을 매우 유연하게 제어하고 자동화하는 방식을 고민해야 한다

Mesh **SFU** **MCU**

- N:M 연결 시 클라이언트 부하가 심해질 경우 성능 저하 우려
- 실시간성 보장
- 실시간성 보장 ❌

> N:M 연결에서 성능 저하가 많이 발생하지 않으며, 실시간성이 보장되는 SFU 방식으로 WebRTC를 구현한다

2️⃣ 오픈 소스

> MediaSoup, Jurento, Janus, OpenVidu

① 오픈 소스 종류

	언어	
Janus	C	
Jitsi Meet		
Mediasoup		
Pion		

3️⃣ 필요한 서버 종류

> 미디어 서버, 시그널링 서버, STUN 서버, TURN 서버

① 미디어 서버

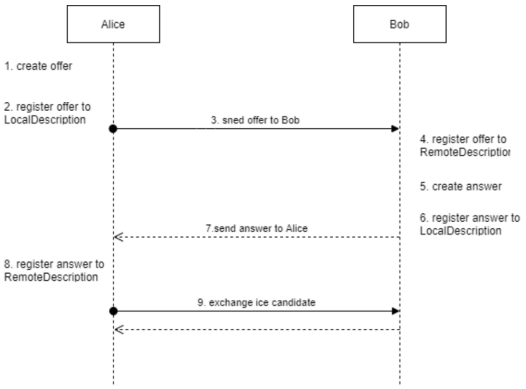
- WebRTC 미디어 서버는

  WebRTC 기반의 미디어 스트림을 중개 및 분배하는 역할을 하는 서버
- SFU, MCU에서는 꼭 필요하다

② 시그널링 서버

시그널링 서버의 프로토콜 : SDP

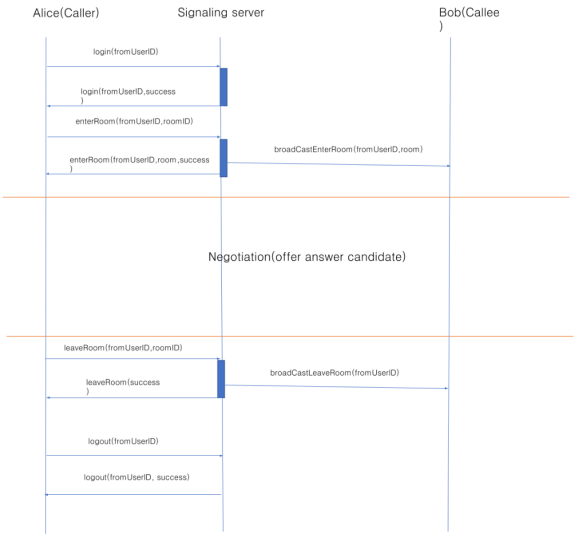
- SDP 프로토콜
- Session Description Protocol
- 제안과 수락 모델 : `Offer/Answer Model` = `Negotiation`



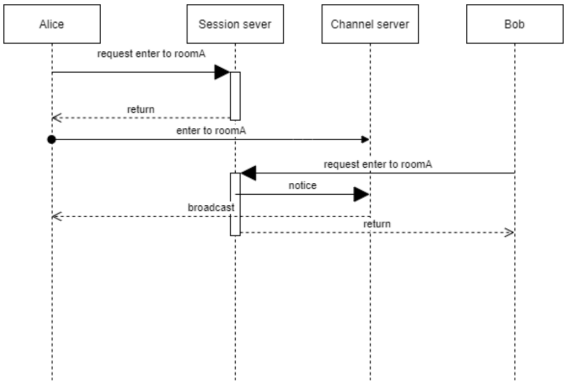
1. **Alice**는 `offer`를 생성한다
2. **Alice**는 생성한 `offer`를 `LocalDescription`에 등록한다
3. **Alice**는 생성한 `offer`를 **Bob**에게 보낸다
4. **Bob**은 받은 `offer`를 `RemoteDescription`에 등록한다
5. **Bob**은 생성된 `answer`를 `LocalDescription`에 등록한다
6. **Bob**은 생성된 `answer`를 **Alice**에게 보낸다
7. **Alice**는 **Bob**에게서 받은 `answer`를 `RemoteDescription`에 등록한다
8. `ICE Candidate`를 교환한다

- 시그널링 서버 = 중계서버
  - offer/answer를 주고받기 위해 필요함
  - 중계를 해야 하기 때문에 양방향 통신 기술을 지원해야 한다
  - `socket.io` 나 `websocket`이 필요함
- 1:1 연결

N:M 연결

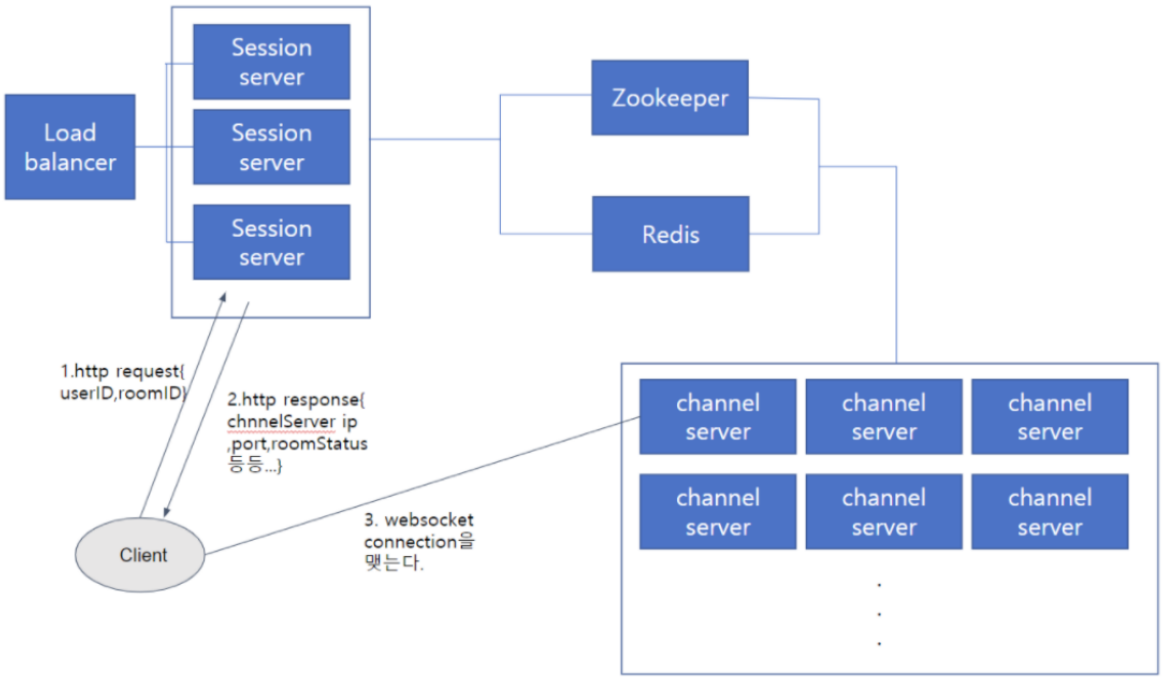


1. [Request] Alice → Signaling Server
   - “Enter to RoomA”
2. [Response] Signaling Server → Alice
   - 성공 여부
   - roomA 정보 : 누가, 몇명, …
3. [Request] Bob → Signaling Server
   - “Enter to RoomA”
4. [Response] Signaling Server → Alice
   - 성공 여부
   - roomA 정보 : 누가, 몇명, …
5. [Negotiation] offer/answer 교환
   - Bob → 방에 있는 모든 사람
   - 방에 있는 모든 사람 → Bob



- `Signaling Server` = `Session Server` + `Channel Server`
  - Session Server
    - HTTP 서버
    - Room 정보 관리
      - 누가 있는지
      - 누가 들어오는지
      - 누가 나가는지
    - Channel Server의 분산처리를 담당
  - Channel Server
    - Websocket 서버
    - offer/answer를 중계함

시그널링 서버 분산처리 : 세션 정보관리



Redis

Zookeeper

- Session 정보 저장 역할
- Session 정보
  - 어떤 room이 있는지
  - 각 room은 어떤 channel server에 접속해 있는지
  - 각 room에 누가 접속했는지
- redis 정규화 :

  (위의 정보를 이용해서 정규화 하면 아래)

  < 데이터 구조 >

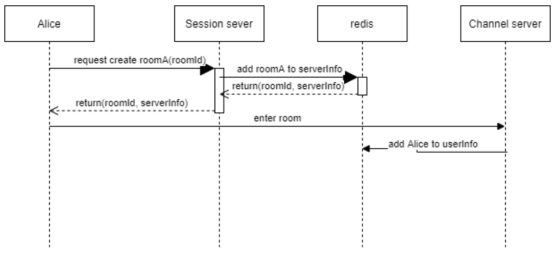
이름	key	value
userInfo	roomId	userIds(list)
serverInfo	roomId	ServerName(String)

  - userInfo : 각 room에 어떤 user가 있는지를 저장하고 있는 key-value
  - serverInfo : 각 room에 어떤 server가 있는지를 저장하고 있는 key-value

- Channel Server의 생사 여부 및 User의 출입 정보를 Session Server에 알려줌

세션 관리 시퀀스 다이어그램

room이 생성될 때



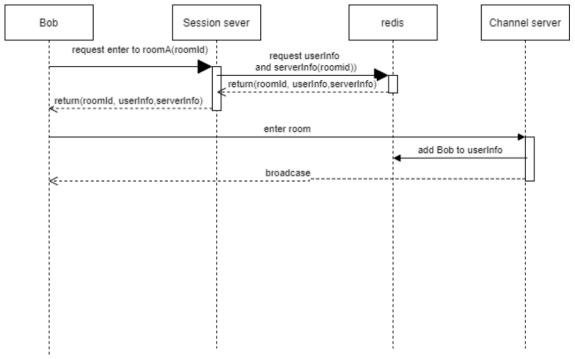
1. [Request] Alice → Session Server

   Create Room (roomId: roomA)
2. Session Server
   - 어떤 Channel이 부하가 가장 적은지 확인
   - Channel Server A로 결정한다
3. Session Server → Redis
   - 데이터 전송
4. Redis
   - serverInfo 데이터 저장
   - key-value : roomA-CServerA
5. [Response] Session Server → Alice

   session 정보 전달
   - return (roomId, userInfo, serverInfo
6. Alice가 roomA에 입장
7. Channel Server → Redis
   - userInfo 데이터 저장 요청

- key-value : roomA-Alice

room에 들어갈 때



1. [Request] Bob → Session Server

   Enter RoomA (roomId: roomA)

2. Session Server → Redis
   - 데이터 조회 (roomId: roomA)
   - userInfo, serverInfo 조회
   - room에 누가 있는지, 어떤 서버에 접속해 있는지 확인
3. Redis → Session Server
   - return (roomId, userInfo, serverInfo)
4. [Response] Session Server
   - 방 정보 전달
   - return (roomId, userInfo, serverInfo)
5. Bob이 roomA에 입장
6. Channel Server → Redis
   - userInfo 데이터 저장 요청
   - key-value : roomA-Bob

③ STUN, TURN 서버

STUN

- Session Traversal Utilities for NAT
- 클라이언트는 자신의 Public IP를 모른다
- 클라이언트가 STUN 서버에 요청을 보내면

  STUN 서버는 클라이언트의 Public IP를 반환한다
- 한계 : 두 클라이언트가 같은 네트워크에 존재할 때
  - Symmetric NAT에서는

    어플리케이션이 달라지면 NAT의 매핑 테이블이 바뀔 수 있기 때문

TURN

- Network Layer IP 주소와 Transport Layer의 UDP 포트 넘버와의 차이를 확인하고

  클라이언트의 Public IP로 응답한다
- 내부 네트워크에 있는 클라이언트의 Private IP로 메세지를 전송한다

STUN

-