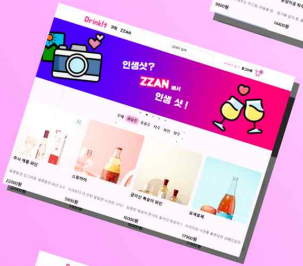
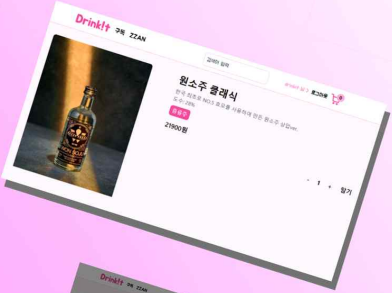
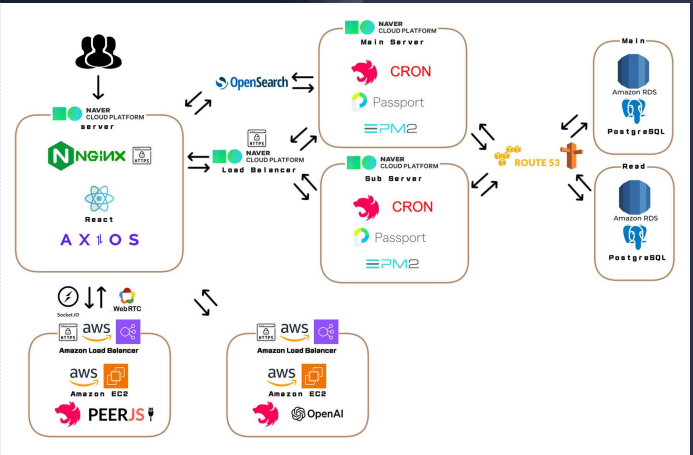




Drink!t

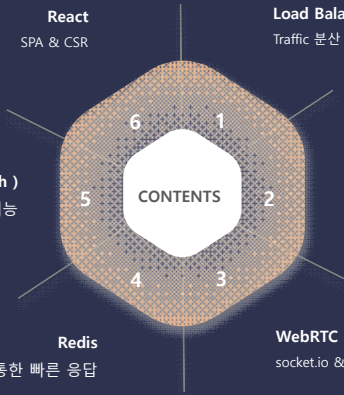
Love, Happiness
Drink!t

Drink!t
Architecture



Stack

Nest.js, TypeORM, Postgresql, Redis
Nclouds, AWS,
ChatGPT, WebRTC, Socket.io,
Opensearch



React
SPA & CSR

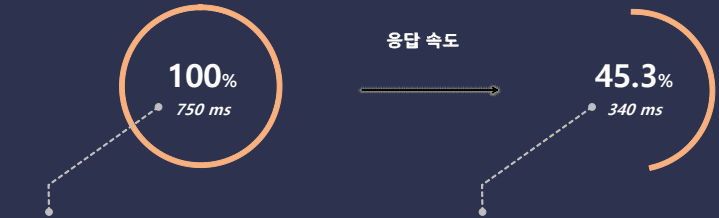
Load Balancer & DB Replica
Traffic 분산 처리

Opensearch (Ellastic Search)
역색인 구조를 활용한 빠른 검색 기능

ChatGPT api
미세 조정 방식 & 체이닝 방식

Redis
비중요 데이터 캐싱 처리를 통한 빠른 응답

WebRTC
socket.io & peer js 를 활용한 화상채팅



Redis, Why?

캐시 데이터 -> 채팅방 목록 & Refresh Token

Database 로부터 응답 받을때의 속도 **750** ms -> Redis 로부터 응답 받을때의 속도 **340** ms

★ 서버 부담 감소 및 UX 의 개선

모든 요청을 서버단에서 처리한다란 말은 즉, 서버에 부하가 증가한다 와 같은 말임으로
많은 GET 요청이 일어날 데이터를 미리 캐싱해두어 분산처리.
인메모리 기반 특징을 활용하여 빠른 응답속도를 통한 UX 의 개선.

원활한 환경을 조성하기 위한 " 선택 "

Ncloud Load Balancer

트래픽 분산 처리

Why ?

● 서버 분리에 대한 생각을 하게 된 이유

1. 프로젝트에 MQ 를 도입한 상황이 아니었습니다.

하나의 서버가 모든 트래픽을 감당하기엔 무리가 있을거라 판단하였고, 회사에서의 상황을 가정해보았을때, 예산은 한정되어있는 상태로 최고의 퍼포먼스를 뽑아내야 한다를 생각했습니다.
팀의 상황에 적용해보았을때, 서버 자체의 성능을 올리는 비용보단 여러개의 프리티어를 사용하는게 효율면에서 뛰어나다 여겨졌습니다.

2. 서버가 다운 혹은 업데이트 해야할경우

서버가 하나였다 가정했을때, 위 상황이 발생되었다 라는 소리는 서비스 중단을 뜻합니다. 안정적인 서비스 제공을 위해서 여러 서버를 운영하는게 맞다 라고 판단되었습니다.

3. 효율적인 Scale-Out 을 위해

만약 하나의 서버에서 모든 서비스 (채팅, ChatGPT, 메인 로직)를 다 담고 있고 ' 채팅 서비스에서 과부하가 생겼다 ' 가정해보았을때 스케일 아웃 진행시 불필요한 ChatpGPT, 메인 로직도 같이 복제가 진행될 테지만 각 서버가 개별의 서비스 로직만을 담당한다면 서버 자체의 자원도 더 효율적으로 사용할 수 있을 뿐더러 다른 서비스까지 신경을 쓰지 않아도 되는 이점을 챙기게 되기에 분리를 진행했습니다.

AI 바텐더

고객센터 자동 응대
간단 질의 응답 서비스

ChatGPT 활용 방식

1. 미세 조정 방식

사람의 언어는 같은 의미의 문장을 다양하게 표현이 가능합니다.
- 택배 언제 도착해?
- 내 물건 언제 와?
- 배송 도착 예정일이 언제야?

결국 하나의 답변을 요구하는 질문들이란걸 알 수 있는데, 이런 다양한 케이스를 학습시킴으로서 올바른 답변을 줄 수 있게끔 하는 방식입니다.

2. 프롬프트 체이닝 방식

단발성으로 ' 질문 - 답 ' 의 형태를 취하는 것이 아닌 사람간의 대화와 같이 진행하는 방식으로, 내가 이전 대화에서 얘기한 요소들에 대해 기억하고 연계할수있는것입니다.
- ' 크림 파스타 추천해줘 ' - > ' 까르보나라를 추천해요 '
- ' 왜 그걸 추천해? ' - > ' 제일 맛있어요 ! '

위와 예시와 같이 그전 질문과 답에 대해 기억하고 연계되어 확인할 수 있는 방식입니다.

각 Server 의 역할

각 로컬 환경을 Peer 라고 칭하겠습니다.

1. STUN Server

STUN 서버의 역할은 Peer가 어떤 IP를 사용하고, 어떤 Port 로 접근을 해야하는지에 대해 알아내는 역할

2. TURN Server

STUN 서버를 통해 교류를 할수 없는 상황일때 중앙에서 미디어 송출을 관리해주는 서버

3. Signaling Server

A, B 라는 Peer 가 있다고 가정을 했을때, 각각의 Peer 는 서로의 SDP를 교환을 해야 합니다. 내가 연결할 Peer의 위치가 어디인지를 모르기때문에 서로간에 위치 전달을 해줄 서버가 필요한데 그 역할을 시그널링 서버가 해줍니다. 혹은 내가 채팅을 전송을 할때 특정 위치에만 전달을 해야한다던가 하는 로직에 대한 처리 또한 가능합니다.

4. ICE Framework

ICE의 역할로는 STUN서버를 통해 연결 후보군들을 관리를 합니다. 하나의 예로 와이파이 연결이었다가 로컬 네트워크 환경으로 변경이 일어날경우, 접속이 끊기게 되는데 이러한 경우 ICE 후보군들중 최적의 경로를 찾아서 재연결을 해주는 역할을 하게 됩니다.

Trouble in WebRTC

◆ Socket 의 연결 끊김 확인

일반적인 유저의 소켓 연결 해제 상황은 다음과 같습니다.

- 브라우저 닫기
- 채팅창 닫기

직접적으로 로그아웃을 통한 연결을 해제하는 상황은 드물기때문에 위와같은 상황속에서도 확실한 소켓 연결 끊김 처리를 원했습니다.

◆ 끊김 처리를 위한 시도

1. 이벤트 기반 메세지 처리

- 'out' 이란 키워드로 메세지가 들어오면 해당 소켓에 대한 연결을 해제해주려 했지만 제시된 위 상황의 경우 메세지 전송 자체가 발생되지 않는것을 확인.

2. Ping Interval 을 통한 확인

- Ping 을 통해 소켓의 연결이 살아있음을 확인하는것 또한 방법이었었지만 그 간격이 좁으면 좁을수록 메인 서버의 부담이 커지는것일것이고 서비스가 확장됨에 따라 비용발생은 더더욱 커질것이기에 배제.

◆ 해결 방법

Nest CLI 를 사용하던 중, gateway를 생성해주는 명령어를 다시 보고 node_modules 내부에 구현되어 있는 Interface 를 확인했습니다.
구현되어 있는 API 들이 존재를 했고, 그중 ' handleDisconnect ' 라는 메서드를 활용하자 앞서 제시된 상황에도 깔끔하게 끊김처리가 가능했습니다.

◆ 하울링 처리

연결이 정상적으로 이어진 후, 내 스피커를 통해 나온 목소리가 다시금 전달 되어 울림 발생되는 문제.

◆ 해결 방법

html 내 등록해준 나의 video 태그의 볼륨값을 0으로 처리.
내 로컬에서 들어간 소리가 다시금 돌고 돌아 전달이 되지 않게끔 처리를 해줌으로써 하울링을 방지할 수 있었고, 또 다른 방법으로는 나의 고유 video 태그에 mute 속성값을 주어서도 해결을 볼수있었습니다.