

ballbot.dev

816万ユーザーの サービスを支えてきた

ソフトウェアエンジニアのキム・スビン
(KIM SUBEEN) です。

歩みを見る

GitHub ↗

Overview

語学・留学

日本語で働ける準備をしてきました

- **JLPT N1** 取得
- **鳥取大学** 交換留学 (2025年度後期)

2 Years

Nudge Healthcare (韓国のヘルスケア企業) のバックエンド

- **19億件のテーブルを** メンテナンス1時間以内にマイグレーション
- **毎分2,500件のリクエストを受ける** サービスのリファクタリング
- **月100万円を超えていた AWS 費用を** 30%削減
- **メモリ 32GB を使い切って停止していた** API サーバーの正常化

4.43 / 4.5

江原大学校 コンピュータ工学科 在学中

- **2027年2月** 卒業予定

2013 ~ 2018

小・中学校

- 2014 **Linux 環境でマイクラフトの24時間サーバーを構築**
VPS サーバーを借りてサーバーを構築し、最大同時接続者数120人を記録しました。
- 2015 **小学校を卒業**
将来の夢に書いたコンピュータープログラマーへの思いは、今も続いています。
- 2015 **タックスパック コミュニティの開発**
PHP で動く少人数グループ向けのサービスを開発し、会員100人ほどを集めました。
- 2017 **スクリーン翻訳機の開発**
スマートフォンを振ると画面の上に翻訳のオーバーレイを表示するアプリを開発し、1,000ダウンロードを記録して収益化も経験しました。

2018 ~ 2021

善隣インターネット 高等学校

ソフトウェア科

- 2018 **モバイルコンテンツコンテストで受賞**
スクリーン翻訳機で、校内大会で2位を受賞しました。

2021
カンウォン
江原大学校
コンピュータ工学科

2021.12 ~ 2023.12
Nudge Healthcare
Backend Engineer
Timespread・Linkareer チーム

- 2019 **VR ゲームの開発**
Unreal Engine を使って、**VRTetris**、**Tooth**、**Unrevived** の VR ゲームを開発しました。
- 2020 **モバイルゲームの開発**
Unreal Engine を使って、**MyRunnerGame** というモバイルゲームを開発しました。
- 2021 **KNU コーディングプラットフォームの開発**
江原大学校 SW 中心大学事業団が進めたコーディングプラットフォームの構築事業で、チームリーダーを務めました。
- 2022 メモリ 32GB を使い切って停止していた API サーバーの正常化
- 2022 月100万円を超えていた AWS 費用を30%削減
- 2023 19億件のテーブルをメンテナンス1時間以内にマイグレーション
- 2023 毎分2,500件のリクエストを受けるサービスのリファクタリング

2024 ~ 現在

カンウォン

江原大学校

コンピュータ工学科

2024

LIKELION 大学 第12期 中央ハッカソン 2位

HEALIXで、韓国の IT 教育企業 LIKELION が主催し、韓国全土の55校から1,500人あまりが集まったハッカソンで受賞しました。

2024

国際交流センターのバディプログラム

日本から来た交換留学生1人を担当して学校生活をサポートし、サークルで良い思い出を作れるよう手助けしました。

2025

国家優秀奨学金（理工系）

韓国で約1,000人しか選ばれない国家優秀奨学金（理工系）を受給しました。

2025

日本・鳥取大学へ交換留学

日本での実際の生活を体験し、茶道部で伝統文化を学びました。

2026

第3回 全国大学ソフトウェア成果共有フォーラム 1位

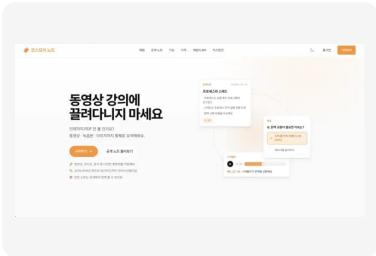
コスモのノートで、韓国全土から13チームが集まった大会で受賞しました。

2026

Align Networks リードエンジニア（契約社員）

engor プロジェクトの開発全体を担当しています。

Projects



2025 ~ 現在

コスモのノート

溜まった講義動画を短時間で消化するための動画要約サービスです。音声だけでなく画面の中のスライドまで読み、サブスクリプションなしで、使った分だけ支払う仕組みにしました。

cosmonote.site ↗

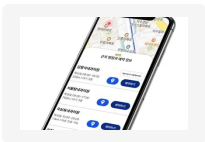


2026 ~ 現在

enqor

リアルタイムのビデオ・音声で、専門家と1対1で相談できるサービスです。秒単位の通話課金機能と、AI エージェントの成果物を、人手をボトルネックにせずに検証するハーネスを作りました。

[App Store](#) ↗ [Google Play](#) ↗

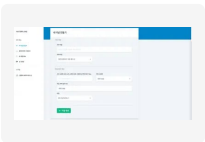


2024

HEALIX

症状をもとに、最も近くて適切な病院をおすすめするサービスです。韓国の公共データポータル上の病院情報とアプリ内の GPS データを、AI に渡すようにしました。

[GitHub](#) ↗



2023

WATERFLAKE

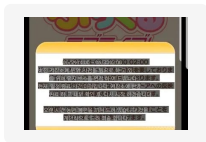
簡単なインストールで、外部からの接続がふさがれたゲームサーバーに、接続できるドメインを発行するサービスです。TCP トンネリングと SRV レコードを中核に使いました。

[waterflake-api](#) ↗

[waterflake-front](#) ↗

[waterflake-tunnel](#) ↗

[waterflake-plugin](#) ↗



2017 ~ 2018

スクリーン翻訳機

スマートフォンを振ると画面をキャプチャし、オーバーレイで訳文を表示します。中学生のときに初めて作り、収益化にも成功しました。

[GitHub](#) ↗

Experience

2021.12 ~ 2023.12

Nudge Healthcare

Backend Engineer

Timespread・Linkareer チーム



19億件のテーブルをメンテナンス1時間以内にマイグレーション

移すデータに優先順位をつけ、いつでも元に戻せるマイグレーションスクリプトを用意しました。



毎分2,500件のリクエストを受けるサービスのリファクタリング

DynamoDB と Redis で設計し直し、6時間に一度更新されていたランキングを、リアルタイムで動くようにしました。



月100万円を超えていた AWS 費用を30%削減

安くて性能の良い Graviton インスタンスに移し、オートスケーリングのルールを組み直しました。



メモリ 32GB を使い切って停止していた API サーバーの正常化

何のエラーも残さずにメモリを使い切ってしまう問題を、本番サーバーのメモリダンプを取って突き止めました。



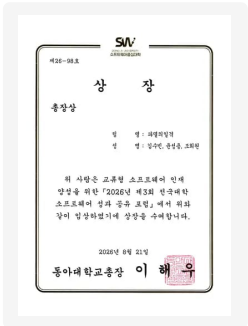
Certificates



2026.02
JLPT N1
国際交流基金



2026.09
情報処理技士
韓国産業人力公団（国家技術資格）



2026
第3回 全国大学ソフトウェア成果共有フォーラム
最優秀賞（全体1位）
東亜大学 総長賞
コスモのノートで、韓国全土から13チームが集まった大会で受賞しました。



2024
LIKELION 大学 第12期 中央ハッカソン
最優秀賞（全体2位）
Younglimwon Soft Lab（韓国のERP 企業）特別賞
HEALIXで、韓国の IT 教育企業LIKELION が主催し、韓国全土の55校から1,500人あまりが集まったハッカソンで受賞しました。



2025
国家優秀奨学金（理工系）
韓国・科学技術情報通信部長官 證書
韓国で約1,000人しか選ばれない国家優秀奨学金（理工系）を受給しました。



2025
Daangn Builder's Camp
Daangn Market 修了証
韓国の地域コミュニティアプリを運営する Daangn（海外名 Karrot）が20人ほどを選抜する少人数制のハッカソンに選ばれました。

2025 ~ 現在

코스모의 노트

動画、PDF などの授業資料の要約サービス

cosmonote.site ↗

参加人数

1人（個人開発）

開発期間

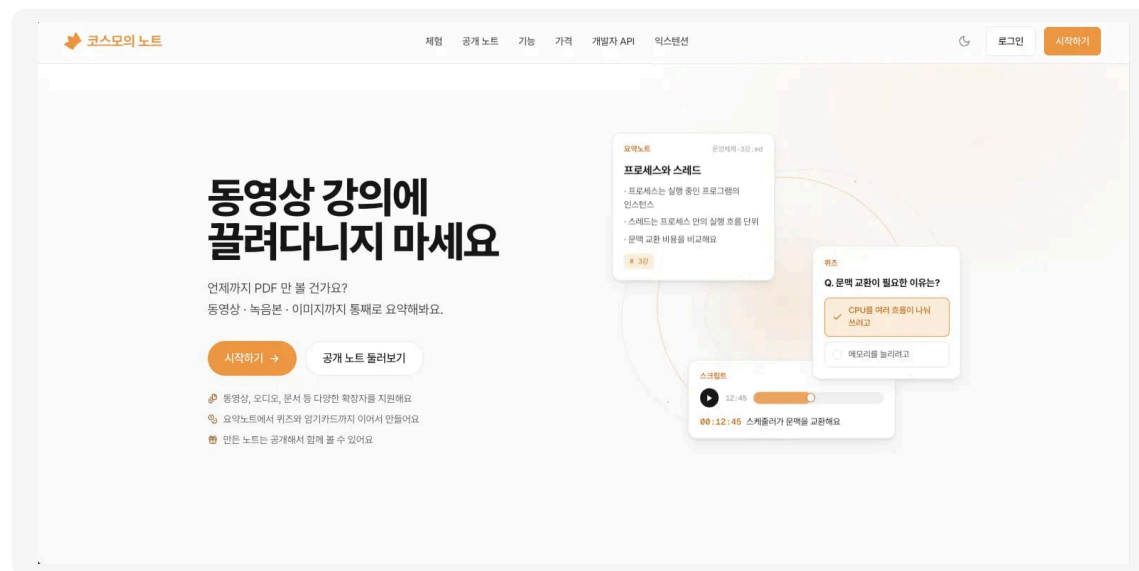
2025.05 ~ 現在

主な技術

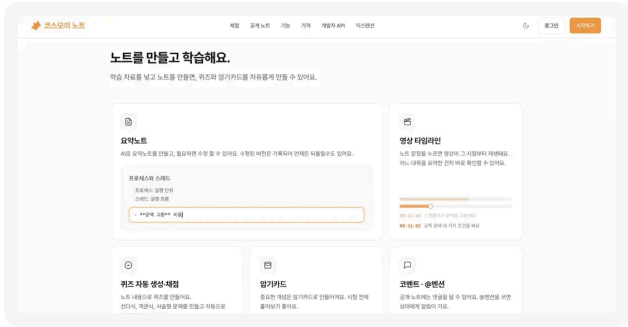
Cloudflare Workers, R2, Containers

受賞

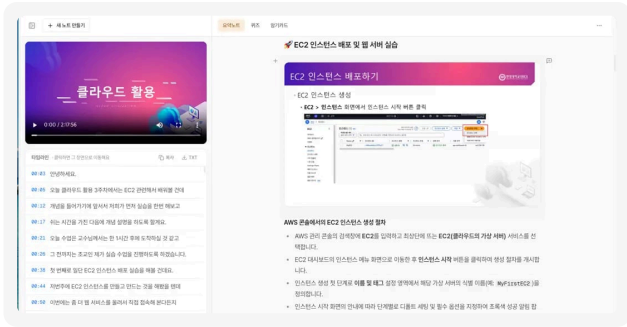
全国大学ソフトウェア成果共有フォーラム（韓国）最優秀賞



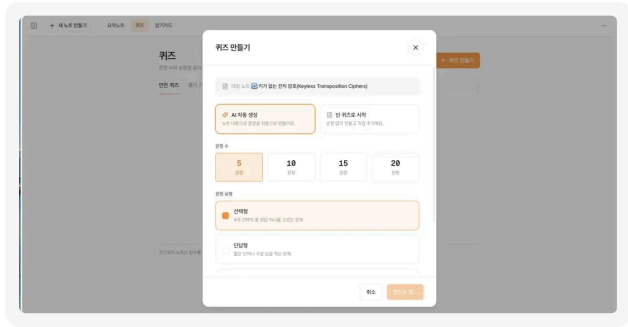
ランディング



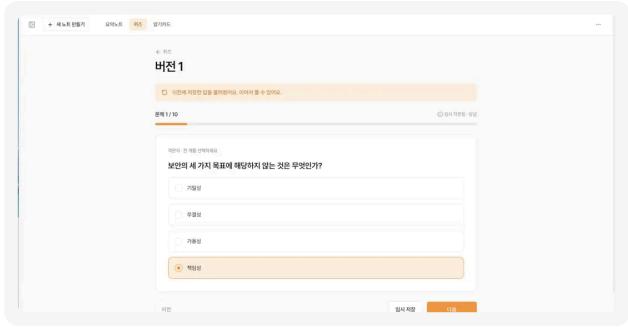
機能



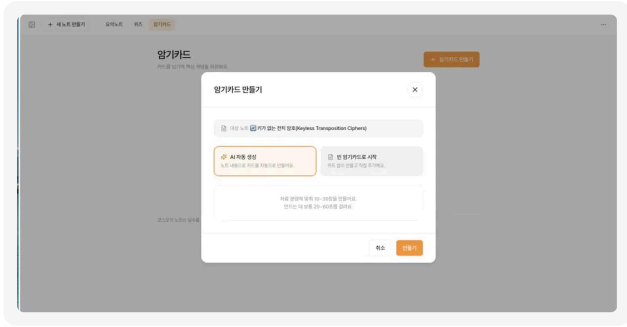
要約ノート



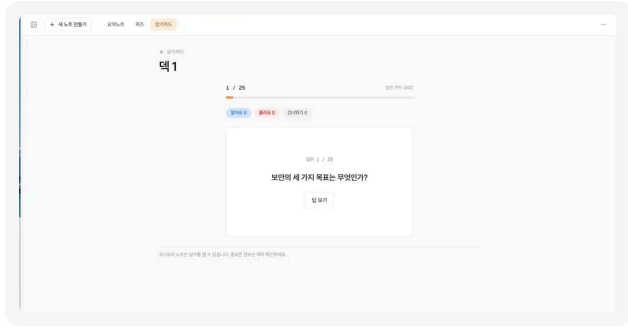
クイズを作る



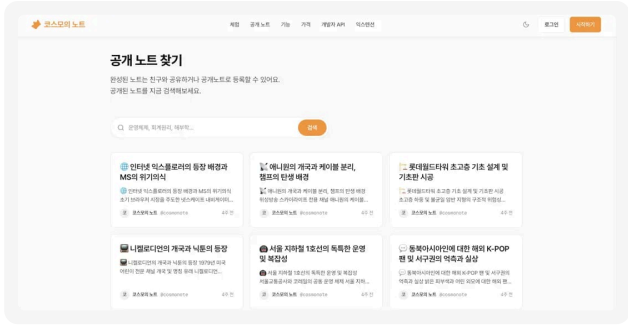
クイズを解く



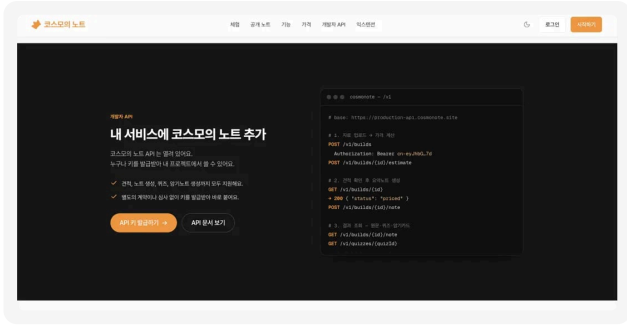
暗記カードを作る



暗記カードで学習



公開ノート



開発者 API

Highlights

- 画面の中のスライドまで読んで、一つのタイムラインにする
- ユーザー1人のために高性能マシンを24時間動かしておかなければならない、というコストの問題を解決する
- フラッグシップモデルを高効率モデルに替えながら、ノートの品質は守り抜く

Motivation

副専攻のクラウド学科は、授業のほとんどが動画講義で行われていました。基礎科目はすでに知っている内容が大半なのに、試験と課題のために、知っている講義を最初から流し直さなければなりませんでした。

しばらくは講義動画をダウンロードして、既存の要約サービスに入れて使っていました。試験の2週間前にまとめて使い、その後2か月は手もつけないのが学生の使い方なのに、月額サブスクリプションなので、使わない日にも料金を払わなければなりませんでした。動画の読み込みから要約までを一度に済ませつつ、従量課金（Pay As You Go）型のサービスを作ってみよう、というのがコスモのノートの始まりでした。

History

2025.05

コスモス ダウンローダー



最初は、講義の視聴画面の右側にダウンロードボタンを付けるブラウザエクステンションを作りました。ボタンを押すとエクステンションがウェブの通信内容を読み、ストリーミングファイルのアドレスをサーバーに送ります。サーバーは .ts (MPEG-TS) 形式のストリーミングファイルをダウンロードして mp4 にエンコードし、ダウンロードリンクを返す、というシンプルな構成でした。

エクステンションを作って公開し、コミュニティに投稿すると、480人以上の学友が「いいね」を押してくれました。ダウンロードしかできないプログラムがこれだけ注目されたのは、この不便さが自分だけの問題ではない証拠でした。

2025.08

要約機能の追加

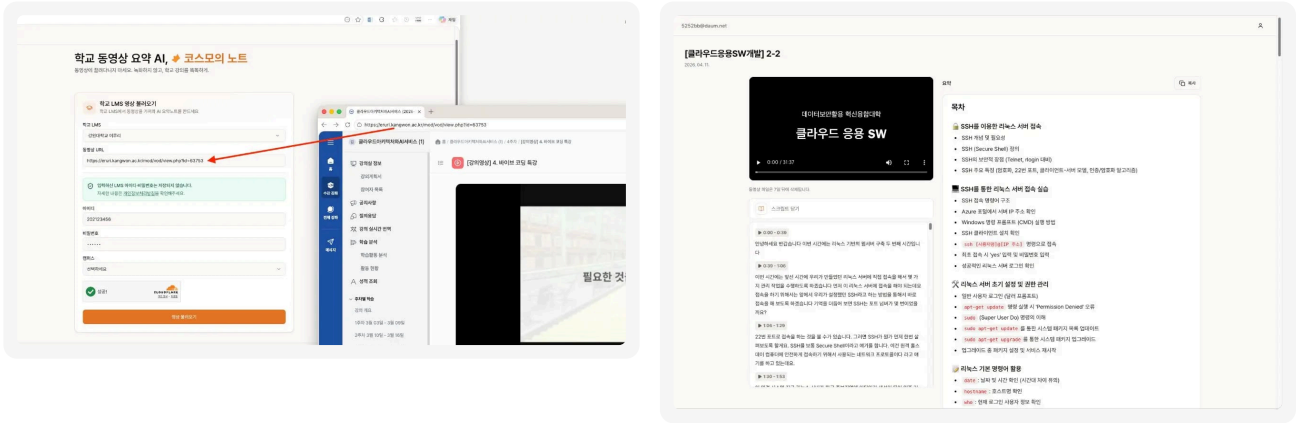


AI ノートの生成機能を付け、ダウンロードボタンを講義のホーム画面に移しました。ボタンを押すと、Cloudflare Browser でサーバーが直接講義サイトにアクセスするように変えました。ログインは、エクステンションが渡したセッションキーをサーバー側のブラウザに埋め込んで回避しました。

エクステンションがしていた仕事を、一つずつサーバーに移していく作業でした。エクステンションをなくし、完全にウェブサービスとして動かすことが目標でした。あわせて決済を連携し、収益化の仕組みも作ってみました。

2026.04

エクステンションの廃止



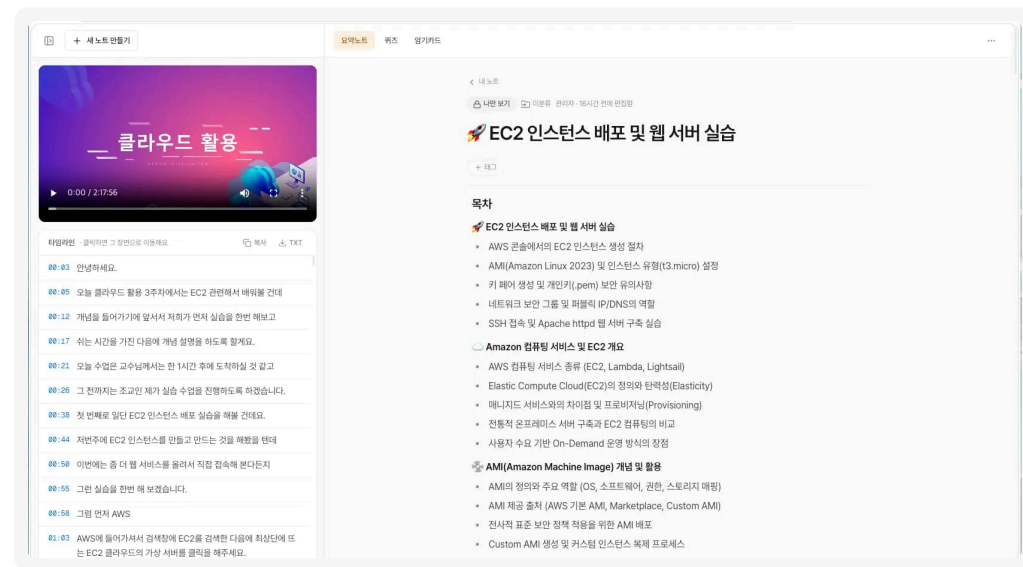
交換留学から戻った後、再びサービスのアップデートを始めました。最大の目標は、ブラウザエクステンションを完全になくすことでした。ブラウザエクステンションはユーザーのブラウザに深くアクセスできる点は良いのですが、モバイル

ではまったく動かず、特定のブラウザに依存するという問題があります。何より、エクステンションを知っている人があまり多くありませんでした。

そこで、ウェブサイトだけで動くように大がかりに作り直しました。学校サイトのログイン情報と講座の URL を受け取ると、Cloudflare Browser で直接ログインし、パケットを検知して動画を見つけ出す仕組みです。ログイン情報は保存しないようにしました。ソフトウェアは複雑になりバグも増えましたが、エクステンションをなくしたことで、使いやすさは比べものにならないほど良くなりました。

2026.09

汎用の動画要約サイトへリニューアル



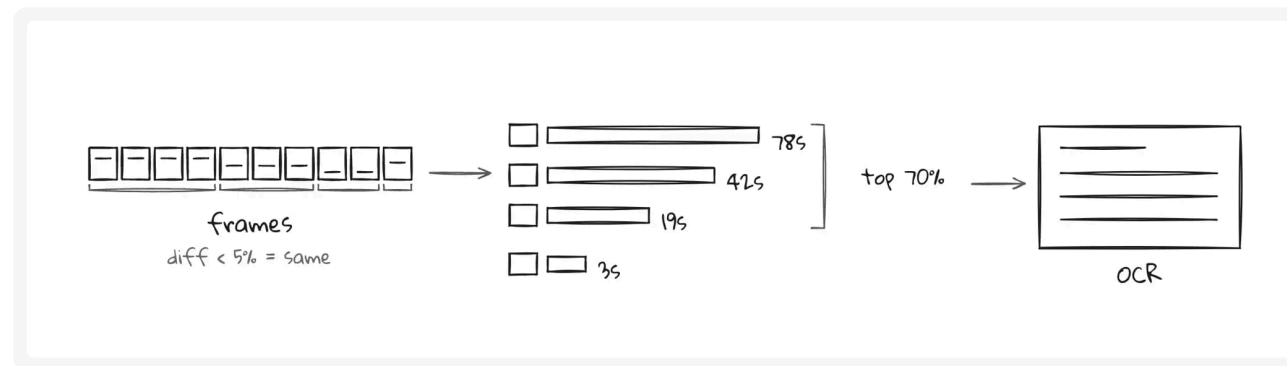
従来の構成では、学校のサイトからしか動画を読み込めませんでした。成長の妨げでもありましたが、著作権のある学校の講義を加工してお金を受け取ることになるため、法的なリスクを避けられませんでした。

売上が出始めたところで嬉しかったのですが、学校サイトとの連携機能は廃止しました。ただ、このサービスのアイデンティティでもあるので完全には捨てきれず、エクステンションに戻しました。動画や録音、画像、文書を直接アップロードする方式にリニューアルし、現在の形になりました。

Architecture

画面の中のスライドまで読んで、一つのタイムラインにする

ほとんどの講義要約サービスは、音声を STT で書き起こしてから、そのテキストを要約します。ところが授業動画で本当に大事なものはプレゼンテーションの中に多くあり、講師がそれを読まずに進んでしまうことも少なくありません。スライドに出てきた数式や図表が、そのまま失われるという問題が起きます。



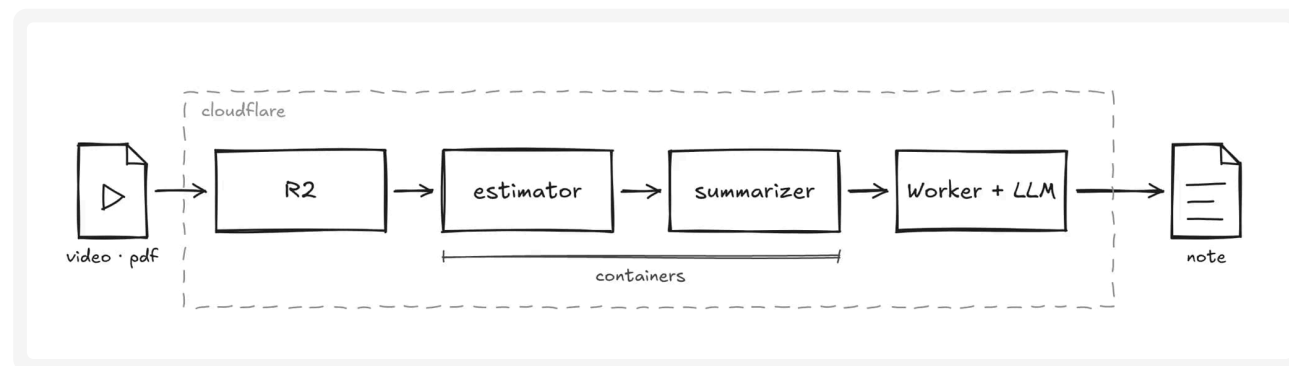
変化のない区間をまとめ、長く映っていた画面だけを読む

そこでコスモのノートには、動画のフレームを分析するロジックを入れました。summarizer は ffmpeg を使ってフレームの変化を追跡します。フレームの変化率が5%未満であれば同じフレームとみなし、同じフレームが長く続いたものをランキングにします。そして、上位70%に当たるフレームを見つけて OCR を行います。

STT の結果と OCR の結果を、一つのタイムラインにマージします。最後にこのタイムラインを Worker に渡し、事前にチューニングしたプロンプトで要約ノートを生成します。

ユーザー1人のために高性能マシンを24時間動かしておかなければならない、というコストの問題を解決する

コスモのノートは ffmpeg で動画のフレームを処理する必要があるため、その瞬間はかなり良いマシンを必要とします。高性能なインスタンスを常時動かしておくと、誰も使っていない時間にも費用がかかります。そこで動画処理の機能にはサーバーレス構成を採用し、費用を大きく削減しました。



アップロードからノート生成までの流れ

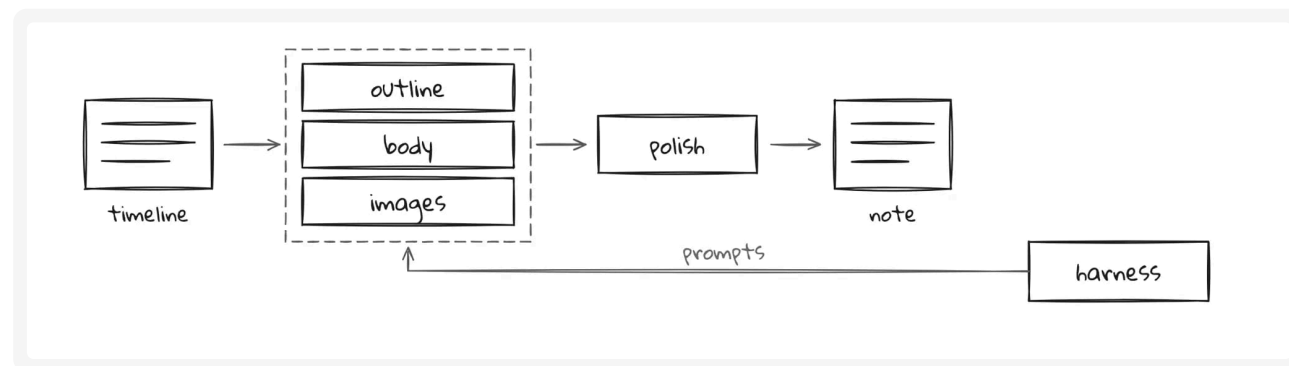
Workers は最大100MB までしかアップロードを受け付けられません。100MB 未満の動画はほとんどないので、サーバーはバケットに直接アップロードできる Presigned URL を発行してクライアントに渡します。ファイルはクライアントから R2 に直接アップロードされます。

アップロードされたファイルは、estimator が検査します。壊れたファイルや不正なファイルでないかを確認し、問題がなければサンプリングして費用を見積もります。ユーザーが見積もられた金額を支払うと、ノートの生成が始まります。見積もりより分析すべき量が多いと損をすることもあります。平均コストに倍率を掛けることで、この問題を最小限に抑えました。

この構成が、そのまま料金プランになりました。2時間の動画で、約130円かかります。他社のサービスが3時間分の動画を分析するのに月790円を求めることを考えると、試験期間にまとめて使う学生にとってはとても合理的な仕組みです。インフラも使った分だけ請求される構成なので、持続可能なシステムです。

フラッグシップモデルを高効率モデルに替えながら、ノート品質は守り抜く

開発初期は、Claude のようなフラッグシップモデルを使っていました。プロンプトのチューニングやタイムラインの整理をあまりしなくても満足できる結果が出ましたが、ノートを1本生成するのにお金がかかりすぎました。2時間の動画を Claude で要約して130円で提供するのは、慈善事業でもなければ成り立ちません。



一度に任せず、分けて任せる

今は Gemini や DeepSeek のような、低コストで高効率のモデルを使っています。その代わり、一度にすべての作業を任せず、目次の生成、本文の生成、画像の配置、推敲などにパイプラインを分けました。小さなモデルは特にコンテキストが大きくなると品質が急激に落ちますが、先に目次を作って1項目ずつ書かせるように仕事を分ければ、品質はそれほど落ちません。また、キャッシングによって費用が一部削減されることもあります。

コンテキストのせいで品質が落ちる問題は解決しましたが、推論能力の差が問題でした。そこで Fable のようなフラッグシップモデルと今のパイプラインの結果を比較するハーネスを作り、定期的に行っています。Fable ほどの性能は出せませんが、コストのわりに良い品質が出ています。

2026 ~ 現在



1:1の専門家相談サービス

[App Store](#) ↗ [Google Play](#) ↗

参加人数

3人

開発期間

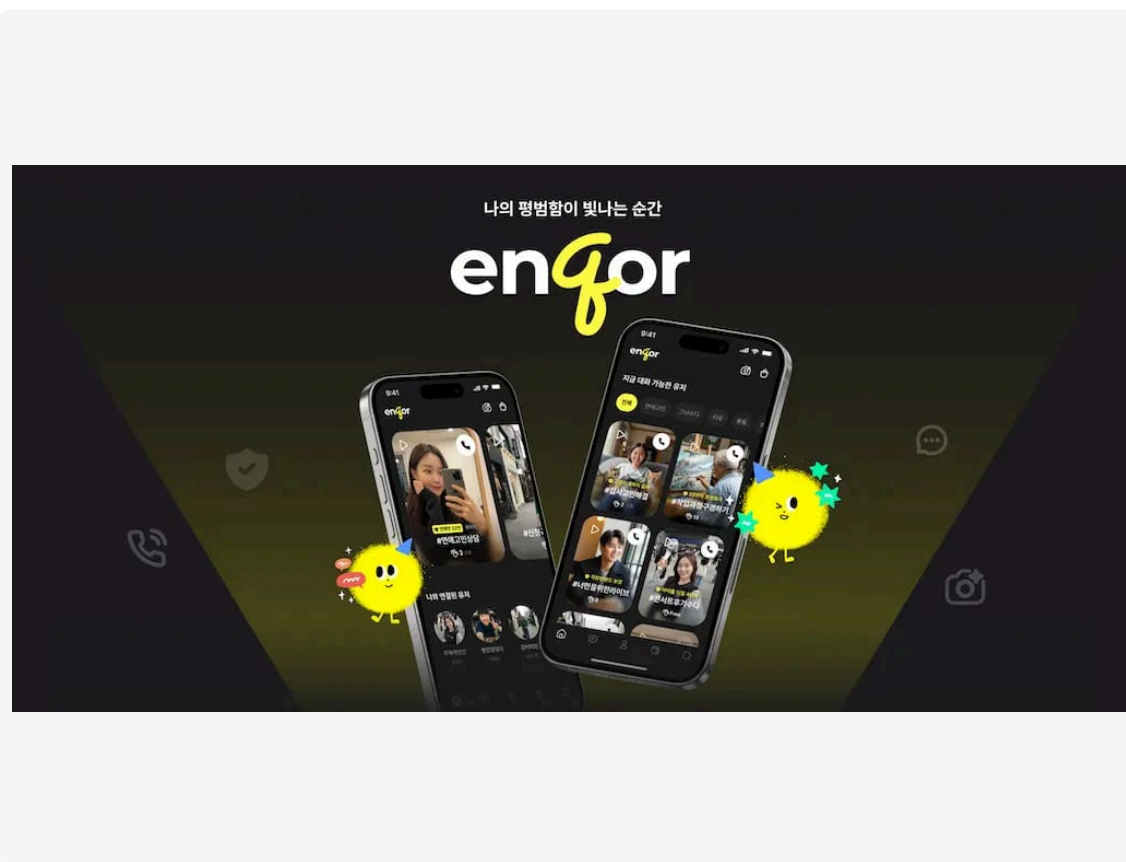
2026.04 ~ 現在

担当

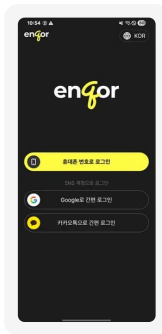
アプリおよびサーバーの開発

主な技術

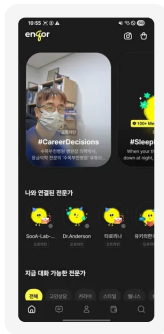
Agora(ストリーミング), NestJS(GraphQL), Expo



enqor



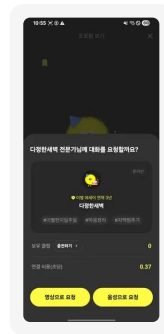
ログイン



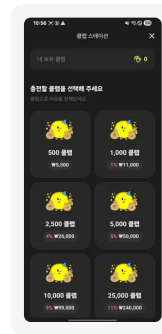
専門家を探す



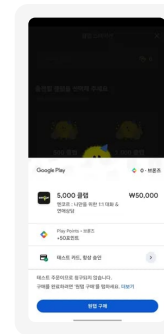
専門家のプロフィール



会話リクエスト



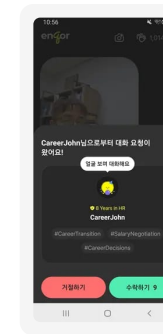
クラップのチャージ



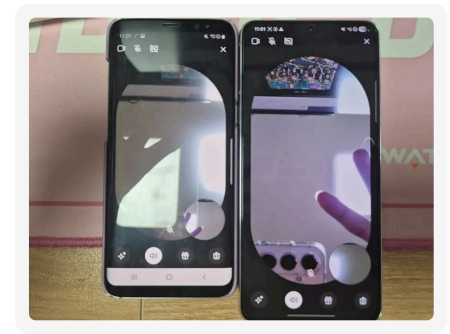
決済



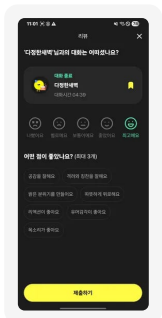
接続中



リクエストの受信



通話



レビュー

Highlights

- 安定して通話がつながっている間だけ、秒単位で課金する
- AI の成果物を、人手をボトルネックにせずに検証する

Motivation

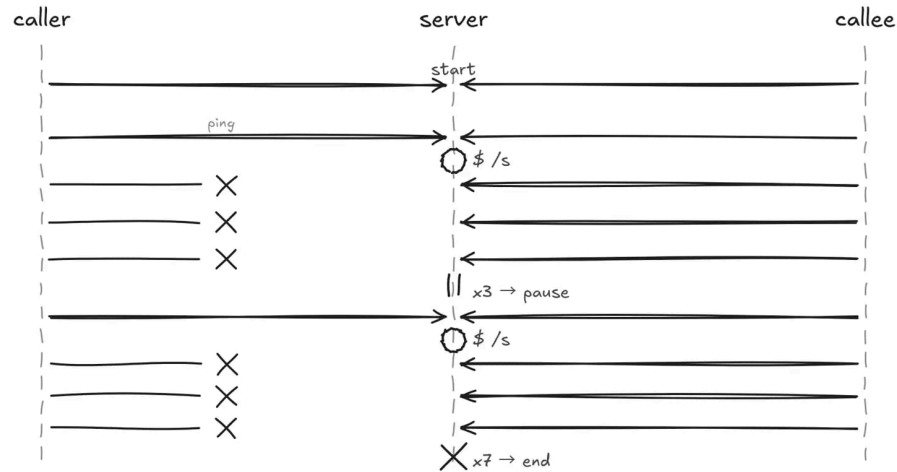
最近は、悩みがあるとまず ChatGPT に聞きます。ところが、いざ心が乱れているときは AI 特有の心のこもらない受け答えにすぐ疲れてしまい、医療や受験のように責任の伴う相談では、AI の答えをそのまま信じるのも難しいものでした。人と話すべき会話は、まだ残っています。

医師のような専門家から恋愛経験の長い人まで、それぞれが自信のある分野で他の人の悩みを聞き、収益を得られる場が必要だと考えました。会話プラットフォームを開発するために Align Networks に参画し、アプリとサーバーの開発を担当しています。ただし、enqor は人が直接プログラミングしたものではありません。企画書とデザインを AI エージェントが読めるようにし、エージェントが出した成果物を検証する環境を作ることが私の仕事でした。

Architecture

安定して通話がつながっている間だけ、秒単位で課金する

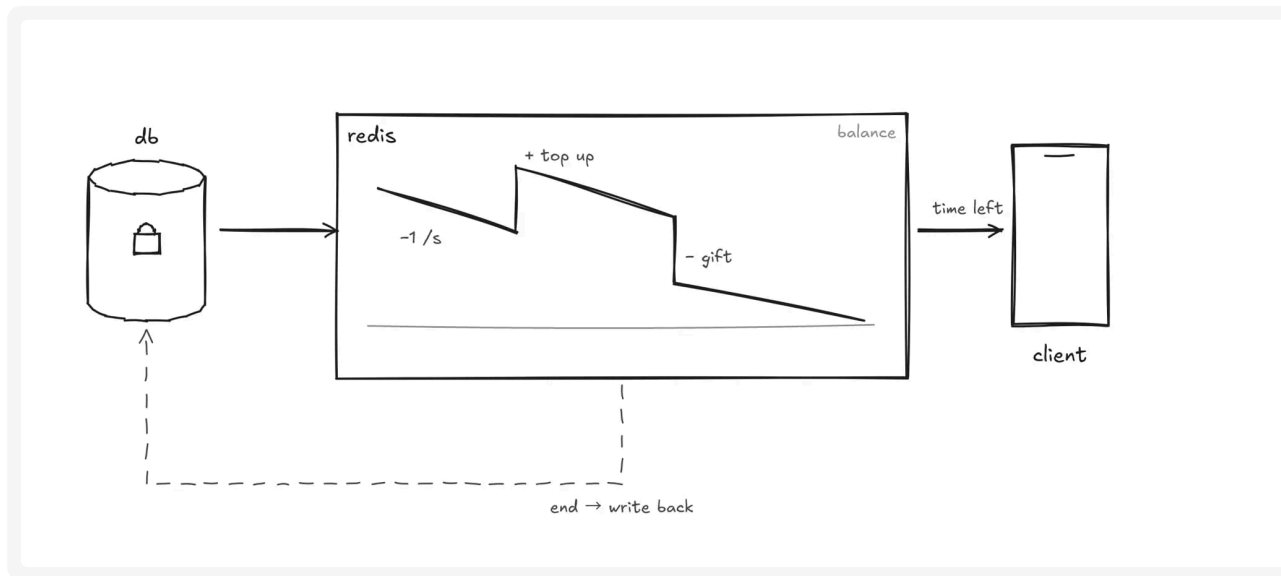
enqor の中核となる機能は、ビデオ・音声通話です。ストリーミングは Agora を使ったので難しくありませんでしたが、課金のパイプラインは自分で作らなければならない課題でした。モバイルは通信状態が悪くなることが多いため、正常につながっていた時間の分だけ課金する必要があり、さらに秒単位の課金まで求められました。



ping で定義した安定した通話

まず、「安定した通話」とは何かを決めなければなりません。これは、サーバーとクライアントがやり取りする ping で定義しました。どちらか一方の参加者でも ping が3回届かなければ接続に問題があると判断し、課金を止めます。7回届かなければ通話が切れたと判断し、セッションを終了するようにしました。

次の問題は、毎秒課金という要件です。最初は、通話時間全体から不安定だった時間を引き、通話が終わった後にデータベースを一度だけ更新するつもりでした。ところが、要件が思った以上に複雑でした。通話中にアプリ内通貨のクラブをチャージできる必要があり、チャージしたクラブで互いにプレゼントもでき、通話可能な時間が20秒未満になれば、もうすぐ通話が終わることも知らせなければなりません。チャージ、消費、照会がすべてリアルタイムで行われる必要があるため、残りのクラブを読み書きの軽いどこかに保存して処理するのが簡単だと判断しました。

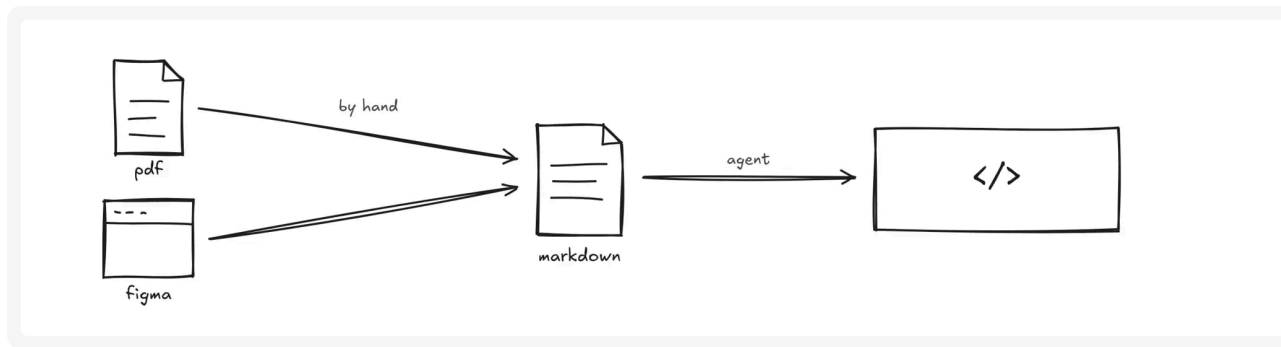


通話セッションの残高

通話が始まると、発信者のクラブ残高を変更できないようにします。現在の残高を読んで Redis の通話セッションに複製しておく、この値が通話中に発信者が使える残高の基準になります。毎秒この値を減らして課金し、チャージやプレゼントがあればこの値を更新し続けます。チャージは決済の記録を残す必要がある、DB にも別途残します。残りの通話時間も、この値をもとに計算してクライアントに返します。通話が終わったら、残高を DB に書き戻します。チャージとプレゼントを除けば、通話1件あたりの DB 操作は開始と終了の2回だけなので、複雑なロジックを直感的に実装できるようになりました。

AI の成果物を、人手をボトルネックにせずに検証する

企画チームが作成したファイルは、PDF の企画書と、Figma で作られたデザインファイルです。人が読んで直すには便利ですが、そのままエージェントに渡すと、どの Figma ノードにどんな説明が入っているのかわかりません。そこで Figma のノードと PDF のページを自分で対応づけて Markdown で書き、ところどころにメモも書き込みました。

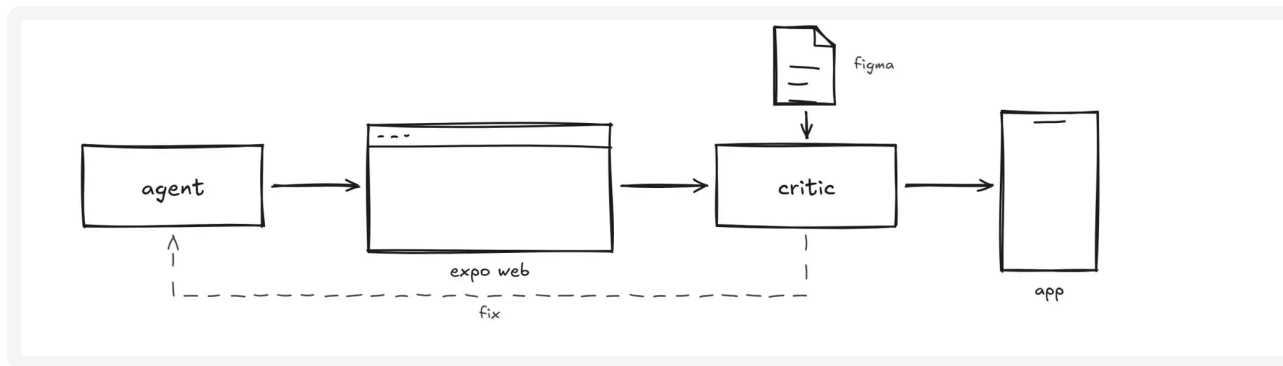


企画書からコードまで

こうして完成した約800行の企画書をもとに、AI エージェントにプロジェクトを最後まで完成させるよう指示しました。30分ほどでプロジェクトが一つできあがりましたが、実行してみると結果はめちゃくちゃでした。ある程度は予想していましたが、起動すらしないとは思わず、なんとか直してログイン画面に入ると、Figma で定義したデザインはまったく反映されていませんでした。ログインのような機能もすべてモックで実装された、まったくのたらいまわしプロジェクトでした。

この問題を顧問に相談したところ、「ハーネスエンジニアリング」という概念を教わりました。ネットで検索してみると、「AI が正しい行動を取れるように環境を作ること」という曖昧な答えしか出てきませんでした。概念を完全には理解できませんでしたが、顧問の助言を受けてフック、批判的レビュアー、テスターを一つずつセットアップしていくうちに、いつの間にか自分だけのハーネスができあがっていました。

ハーネスで一番難しいのは、「成功条件」を定義することです。たとえば Figma で定義されたデザインどおりに画面を実装できたかを判定するには、そのデザインと実装された画面が一致しているかを評価するロジックが必要です。最初はアプリの画面をキャプチャして似ているかどうかだけを調べ、独立したサブエージェントがこの比較を担当しました。悪くはありませんでしたが、良い成果物とも言えませんでした。あちこちでアイコンが違い、テキストの位置がずれていました。何より、アプリの画面をキャプチャして Figma と照らし合わせる作業がとても遅く、トークンを大量に消費しました。



先にウェブで画面を組むハーネス

この問題は expo web で解決しました。まず実装したい画面を expo web で動かせば、ヘッドレスブラウザを開いて実装の状態を難なく確認できます。ウィンドウサイズを変えるだけで、モバイル・タブレットの解像度で崩れる部分がないかもすぐにわかります。こうしてウェブでの実装が終わったコードをアプリに移す順番に変わると、評価のコストが大きく下がりました。Figma とウェブ上の画面がどれだけ似ているかは批判的エージェントが判定しますが、アイコンを自分で生成できないようにルールを入れ、Figma MCP に問題が起きて動かない場合はハーネスがそもそも止まるようにする preflight の手順も追加しました。その結果、シンプルな画面はほぼ完璧なレベルで、複雑な画面も70%ほどの完成度で仕上がりました。

Retrospective

今回の経験で、AI に良い資料と環境を与えれば、それに見合う結果が出るとわかりました。そして AI が生み出した結果も、「どう」検証するかより「どこまで」検証するかを考えるほうが正しいと感じました。

金融、宇宙産業、防衛産業のように、一度の事故が惨事になるドメインでは、すべてのコードを隅々まで検証しなければならないでしょう。しかし今回のプロジェクトのように、ほとんどはそうではありません。開発者は、AI が今何をしていて、この作業が終わったらどこまで検証すべきで、どのファイルが変わることを期待しているのか、その程度を把握していれば十分だと考えています。その中でも、AI がしているのが単純な作業であれば何度か触ってみてテストを終えることもできますし、中核のロジックであれば設計まで細かく議論する必要があります。

開発者がコードを書く人から、判断して責任を負う人へと変わったことを、このプロジェクトで確かめました。

2024

HEALIX

症状にもとづく病院探しサービス

GitHub ↗

参加人数

6人

開発期間

2024.07 ~ 2024.08

担当

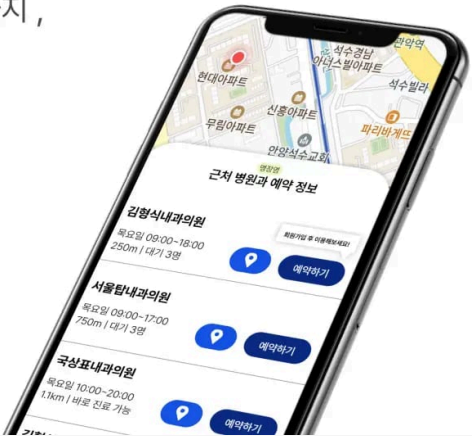
バックエンドリード

主な技術

ChatGPT API, NestJS

증상 분석부터 나와 가까운 병원까지,
HEALIX

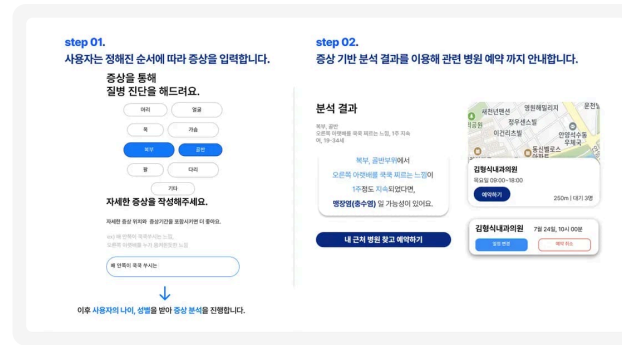
Zeegle
김수빈 김원호 김재연 이진호 조준환 최혜선



HEALIX



企画の背景



症状の入力と病院の予約

Motivation

人は体のどこかが痛いとき、どの病院に行けばいいのかわからないことがよくあります。すぐに思いつくだけでも、熱の出る風邪をひいたら内科に行くべきか耳鼻咽喉科に行くべきかわかりませんし、お尻のような馴染みのない部位が痛くなってくると、その悩みはさらに大きくなります。以前は、ネットで検索して適切な診療科を探し、地図で病院を探さなければなりませんでした。HEALIX は、この不便さを解決するために開発されました。

Video



HEALIX のデモ動画

<https://www.youtube.com/watch?v=p3cPmhgZELg>

2023

WATERFLAKE

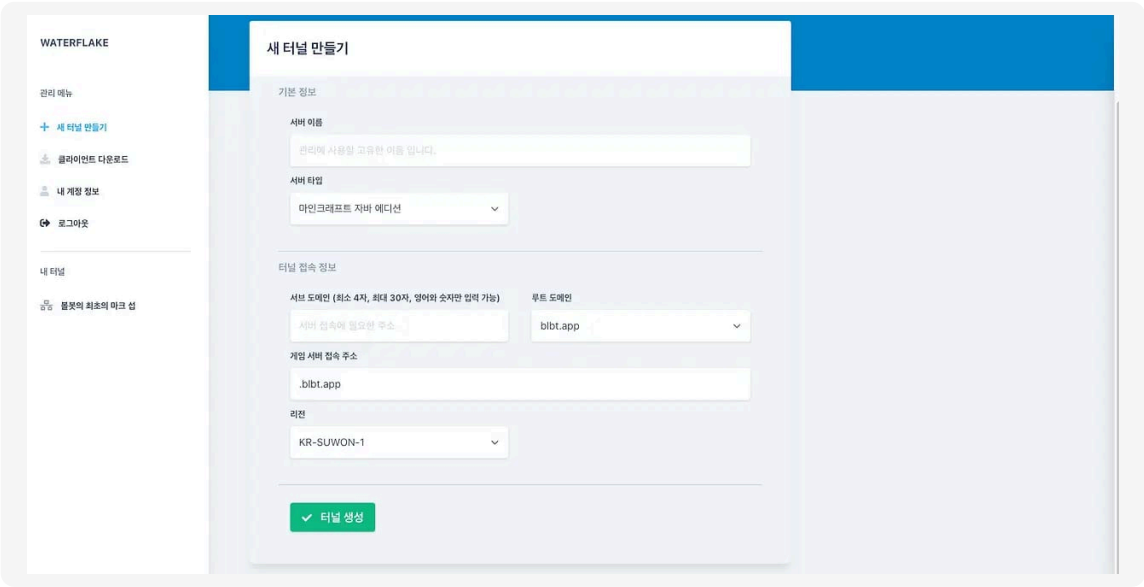
簡単なインストールだけで、マイクラフト
サーバーにドメインで接続できるようにする

[waterflake-api](#) ↗ [waterflake-front](#) ↗
[waterflake-tunnel](#) ↗ [waterflake-plugin](#) ↗

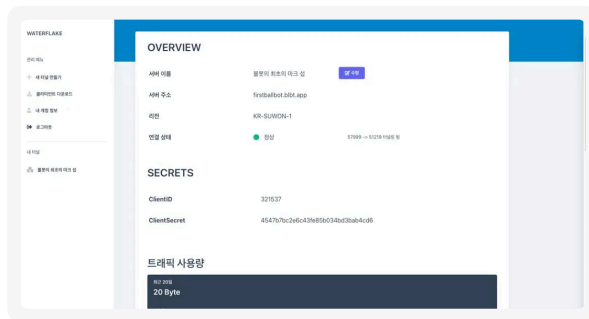
参加人数
1人（個人開発）

開発期間
2023.03 ~ 2023.05

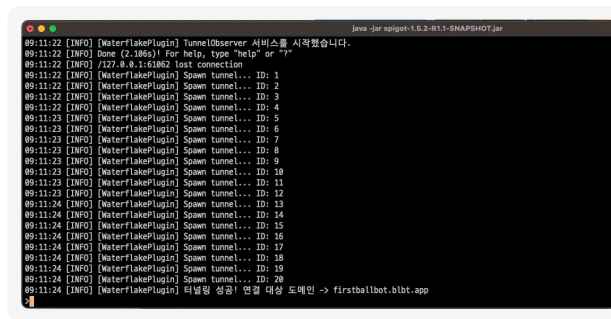
主な技術
NestJS, Java



トンネルを作る



トンネルのダッシュボード



サーバー接続

Highlights

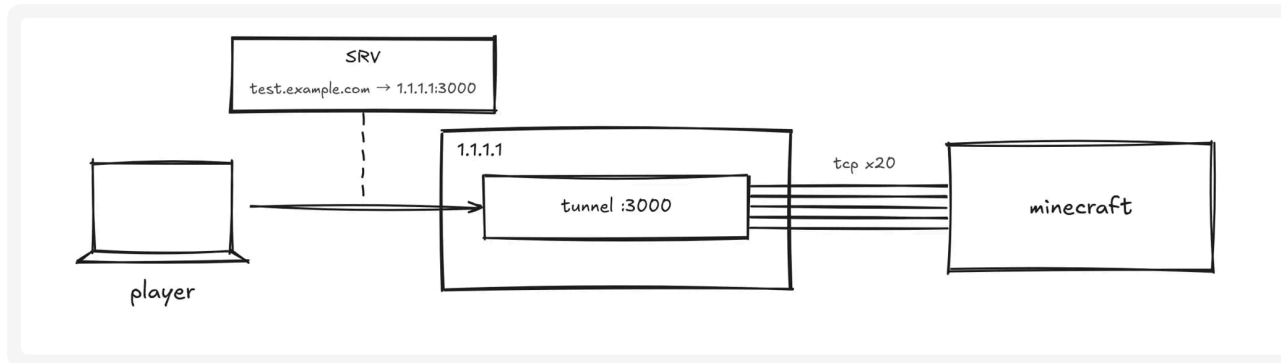
- SRV レコードで、サブドメインが特定の IP のポートを指すよう自動で設定する
- ゲームサーバーとトンネリングサーバーの間を、TCP ソケットでトンネリングする

Motivation

10年ほど前、マイクラフトというゲームにはまり、みんなで遊べるサーバーを立てることが一番の目標でした。コンピュータのことなどまったく知らない小学生が、ブログの記事だけを読んで VPS サーバーを借り、Linux 環境でゲームサーバーを動かしました。そして複雑なネットワークの設定までこなして、ついにサーバーを公開しました。そのときの達成感は人生で最も大きく、決して忘れられない思い出です。そして今、「もし複雑な手順なしに、簡単にゲームサーバーを公開できるツールがあったらどうだろう？」という考えから始めました。

Architecture

SRV レコードで、サブドメインが特定の IP のポートを指すよう自動で設定する



ドメインからゲームサーバーまで

ユーザーが WATERFLAKE のウェブ画面でトンネルを作成すると、Cloudflare API を通じて SRV レコードを作成します。そして、トンネリングを担当する仮想マシン上に、ランダムなポートでトンネリングサーバーを立ち上げます。トンネリングサーバーは2つの接続を受けます。1つは SRV レコードを通じて入ってくる接続、もう1つはマイクラフトのサーバー側へ出ていく接続です。この2つの接続をパイプでつなぐことで、外部から入ってくるパケットが自然にマイクラフトのサーバーへ流れていきます。ユーザーには、まるでドメインのアドレスを入力すれば接続できるサーバーのように見えます。

ゲームサーバーとトンネリングサーバーの間を、TCP ソケットでトンネリングする

トンネリングサーバーとゲームサーバーの間の TCP トンネルは、20個のソケットで管理されます。プレイヤーがサーバーに接続すると、このソケットのうち1つを占有します。プレイヤーがゲームサーバーから抜けると、そのソケットは破棄され、新しいソケットが作られます。このサイクルを繰り返して、ソケットの数を管理します。幸い、マイクラフトのサーバー設定には最大接続人数が決まっているため、この値の分だけソケットを管理すれば、運用に問題はありませんでした。

19億件のテーブルをメンテナンス1時間以内にマイグレーション

背景

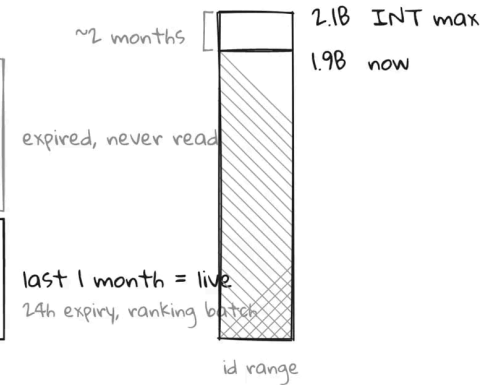
Timespread には、ランダムボックスという機能があります。広告を見るとアプリ内通貨の「キャッシュ」がランダムに入ったボックスがもらえ、そのボックスを開けるとキャッシュが入るというシンプルな機能でした。ボックス1つは、luckybox テーブルの1行でした。ユーザーが増えるにつれてこのテーブルに19億件が積み上がりましたが、主キーがINT で宣言されていたため、上限の21億まで2か月ほどしか残っていない状態でした。このテーブルを、BIGINT で宣言した新しいテーブルに移す必要がありました。

1回目の試み

まず、サービスの特徴を調べました。ボックスは24時間が過ぎると期限切れになり、ランキングを処理するバッチプログラムは直近1か月のデータしか読んでいませんでした。メンテナンス時間内にこのデータだけを移せば、サービスを再開できるとわかりました。

luckybox 1.98 rows

id INT	cash	created_at
1	300	2021-12-01
2	100	2021-12-01
...	...	...
...	...	...
1,899,999,998	500	2023-06-19
1,899,999,999	1,000	2023-06-30
1,900,000,000	300	2023-06-30



19億件のうち、実際に読まれるのは直近1か月分だけ

チームリーダーと相談し、平日の明け方にサービスを1時間止めて、あらかじめ用意したマイグレーションスクリプトを実行することにしました。スクリプトは、次の作業を順番に行います。

- 既存テーブルの名前を RENAME して、接続を切ります。
- 新しいテーブルを BIGINT で作り直します。
- 万一の重複を防ぐため、PK が22億から始まるようにします。
- 1か月前のデータが何番の ID なのかを調べます。
- MySQL のプロシージャで、100件ずつバッチでデータをコピーします。

スクリプトを書くうえで最も重視したのは、「ロールバックできるか」でした。作業の途中で問題が起きたらすぐに止められなければならない、進捗も目で確認できる必要がありました。データの挿入はクエリ1行でも書けましたが、進捗がわからず途中で止めることもできないため、作業を細かく分けることにしました。

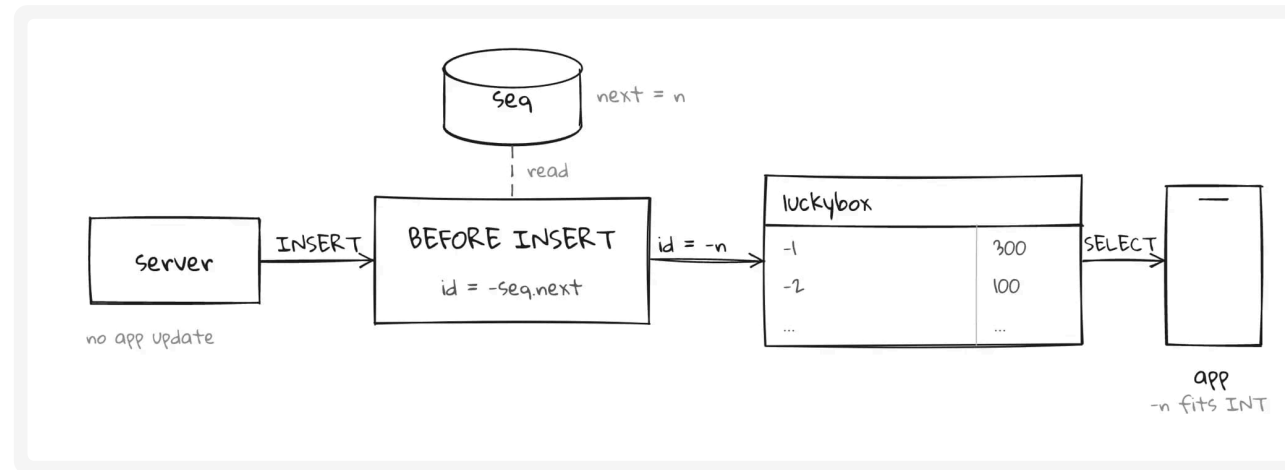
マイグレーションは無事に進みました。1か月分のデータが入り、新しいボックスは22億から作られ始めました。ところが、なぜか Android アプリでクラッシュが起き始めました。残りのメンテナンス時間が少なかったため、用意しておいた手順どおりにロールバックし、サービスは元の状態に戻りました。

原因

理由は単純でした。アプリでも ID を INT で扱っていたため、21億を超える値を処理できなかったのです。サーバーだけ直せば済む問題ではありませんでした。別の方法を探す必要がありました。

2回目の試み

チームリーダーと解決策を話し合っているうちに、ふと INT には負の領域もあることを思い出しました。幸い unsigned で宣言されていなかったため、PK を負数で保存する方法について話し合いました。チームリーダーも良い方法だと言ってくれたので、すぐに作業に取りかかりました。



トリガーが PK を負数に差し替える流れ

BEFORE INSERT トリガーを設定し、値が入ってくるときにシーケンステーブルから次の値を読んで、PK を負数に替えて入れるようにしました。テストサーバーで、サーバーとアプリの両方が負の PK を問題なく処理できることを確認し、明け方にトリガーを設置しました。

その日以降に挿入されるすべてのデータの PK は負数で入り、アプリでも問題は起きませんでした。この対応で目の前の障害を防ぎ、約2年分のリファクタリングの時間を稼ぎました。

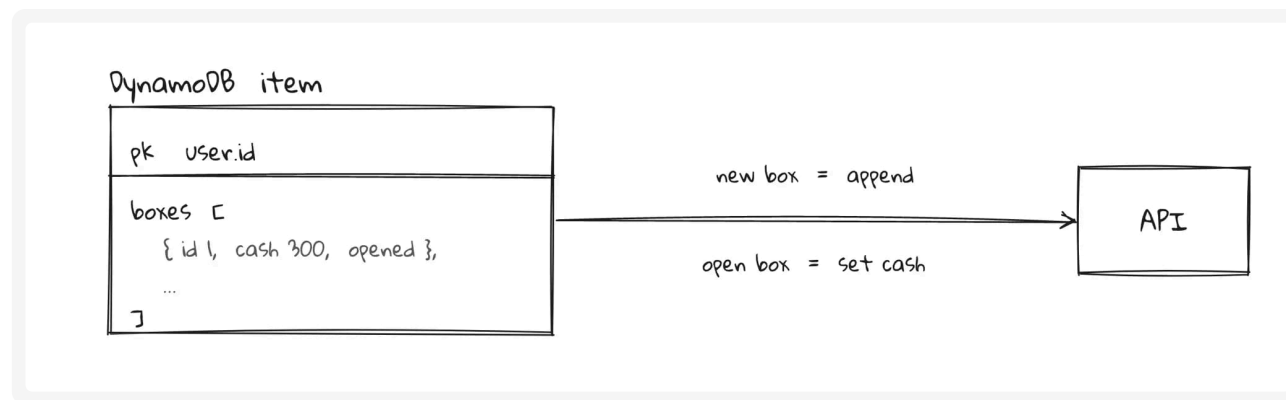
毎分2,500件のリクエストを受けるサービスのリファクタリング

背景

先のマイグレーションで時間を稼いだ直後から、次の作業の準備に入りました。まず、アプリで ID を整数として扱っていたのを String に変え、そのリリースがユーザーの99%に行き渡るまで待ちました。ちょうど運営チームから、ランダムボックスの確率調整機能がわかりにくいのでリニューアルしてほしいという要望があり、代表の承認も得られたので、ようやくリファクタリングに着手できました。ボックスサービスの /luckybox エンドポイントは、当時毎分2,500件前後のリクエストを受けていました。

ボックスをどこに保存するか

まず、ボックスをどこに保存するかを考えました。以前の設計ではボックスをすべて MySQL に保存していたため、ボックスを作るたびに INSERT が、開けるたびに UPDATE が発生していました。これを DynamoDB に移し、パーティションキーで読み書きする速度が速いという長所を、最大限に活かすことにしました。



1つのパーティションキーに、ボックスを配列で

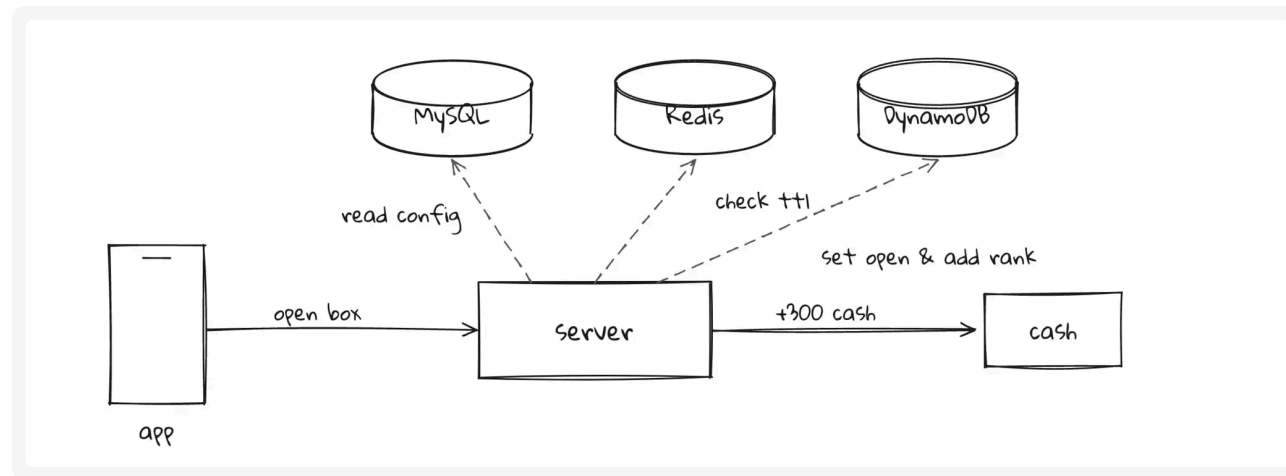
ボックスができれば配列に値を追加し、ボックスを開けたらその項目だけを直せば済みました。見た目は MySQL と変わりませんが、1つのパーティションキーで照会・更新するので、はるかに速くなりました。また、ユーザーがその日初めてアクセスしたときに溜まっていたデータを消すようにして、以前のように値が際限なく増え続けることも防ぎました。

新たにできたポリシーで、1人のユーザーが開けられるボックスは最大20個までという制約もありました。この制約もボックスの length を数えるだけですっきり解決でき、さらに効率的でした。

ランキングと設定はどこに置くか

ランキングシステムも新しく作りました。Redis の ZSET は重複のないキーをスコア順に並べておき、順位の読み取りと更新を $O(\log n)$ の時間計算量で行えます。さらに、高額当選者が一定期間ふたたび当選できないようにする必要があるという要件もありましたが、当選したユーザーの ID に TTL をかけて Redis に保存しておけば、データベースで当選記録を照会しなくても簡単に解決できました。

当選確率のような設定値は、MySQL に残しました。管理者コンソールとの連携を考慮する必要があったからです。ボックスを開けるたびに MySQL を一度読まなければならないという短所はありますが、設定をリアルタイムに反映でき、読み取りだけなので大きな負担にはならないと判断しました。



ボックスを開けるときに、3つのストアを回る順序

結果

完成したシステムは、少しずつトラフィックを受けていきました。朝は10%ほどだけ受けながら様子を見て、問題がなかったのが午後100%まで上げました。MySQLの負荷が下がり、何よりランキングに活気が出ました。6時間に一度更新されていた値が、リアルタイムで動くようになったからです。運営チームも、当選確率を設定しやすくなったと満足していました。

問題を見つけてチームリーダーに報告した日から最後のリファクタリングまで、まる1年かかりました。自分の書いたコードが少しずつトラフィックを受け、100%までリリースされていく様子を見守った経験は、今でも忘れられない開発の楽しさです。

月100万円を超えていた AWS 費用を30%削減

背景

Timespread は、サーバー費用として毎月100万円を超える金額を使っていました。チームリーダーから、この費用を減らす方法を探してみようと言われました。

どこにお金がかかっているのか

Cost Management で確認すると、費用が最もかかっているのは3か所でした。

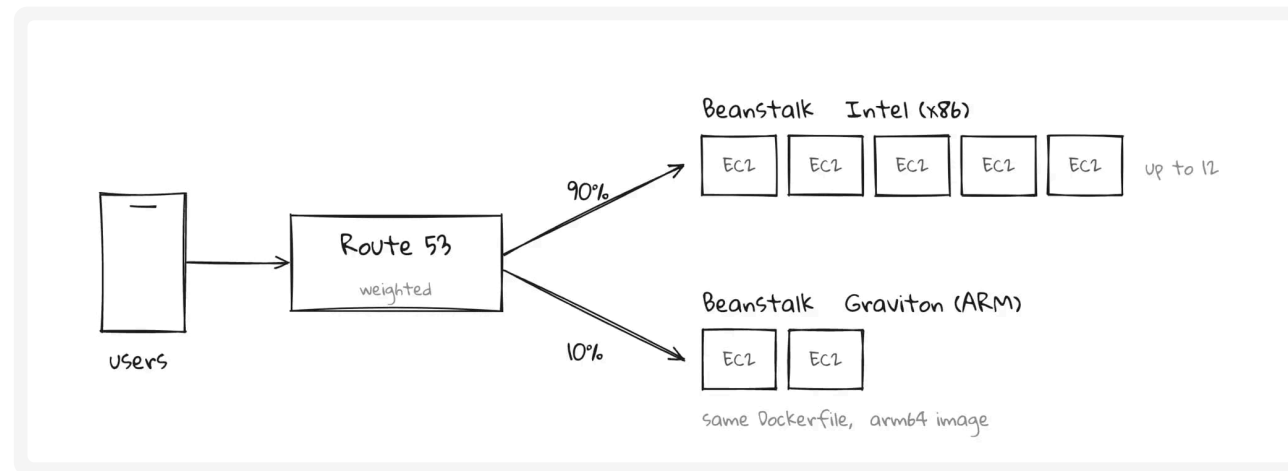
- RDS
- Beanstalk (EC2)
- ネットワークトラフィック

RDS はすでに負荷が高く手を入れる余地がなかったので、EC2 で減らせるところを探しました。ちょうど ARM 系の Graviton インスタンスが新しく登場し、同じ性能を20%ほど安く出せるという AWS の発表を見て、自分たちのサービスに適用してみることにしました。

Graviton に移す

まず Graviton インスタンスを1台作り、サーバーを立ち上げてみました。OS のバージョンとアーキテクチャが変わったため、従来の方法では起動しませんでした。幸い Dockerfile でビルドすると ARM アーキテクチャでも安定して動き

ました。そこで既存の CI/CD パイプラインを直して GitHub Actions でクロスビルドにより ARM イメージを作るようにし、Graviton インスタンスで動く Beanstalk を新たに作って、そのイメージを使うようにしました。



Route 53 の加重ルーティングで、トラフィックの10%だけを新しいインスタンスへ

テストサーバーで QA チームと一緒にサーバーが正しく動くかを確認したところ、何の問題もありませんでした。Route 53 の加重ルーティングで新しいサーバーにトラフィックの10%だけを流し、大きな問題が出なかったので、最終的にすべてのトラフィックを Graviton インスタンスが受けるようにしました。

オートスケーリングのルールを組み直す

数日後には、CPU 使用率のデータを見て、時間帯ごとのスケールアウトのルールを調整しました。ピーク時に最大12台まで増えていたインスタンスを8台に減らし、夜間は1台だけ動くようにしました。インスタンスの切り替えとスケールアップルールの調整を合わせて、最終的に費用を30%ほど削減できました。

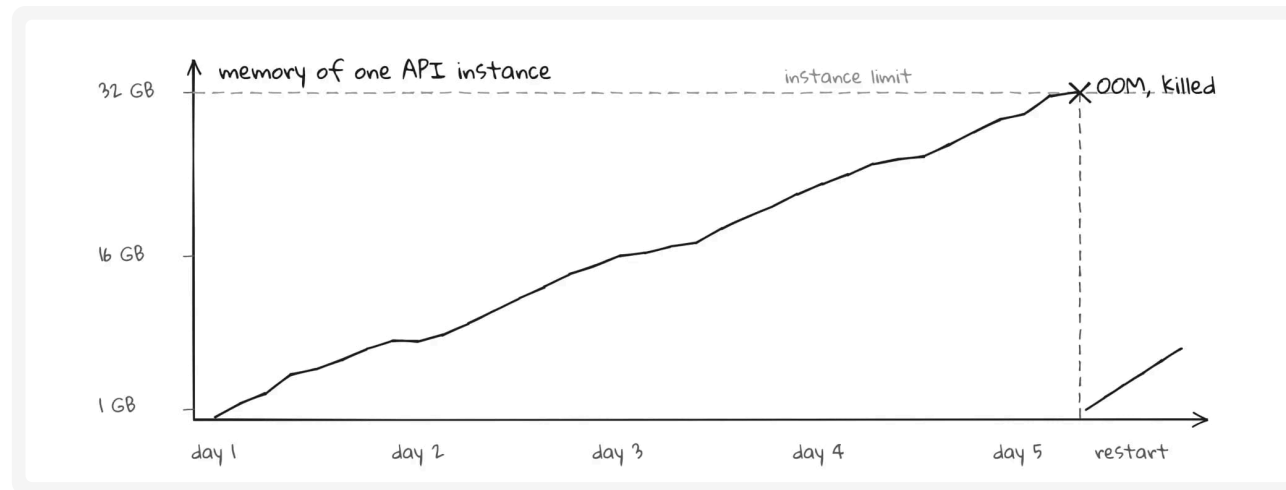
メモリ 32GB を使い切って停止していた API サーバーの 正常化

背景

入社して4か月ほど経った頃のことでした。マシンごとのメモリ使用量が、不自然なほどばらばらでした。あるマシンは20GB 以上を使い、あるマシンは1GB ほどしか使っていませんでした。そして、ほぼすべてのマシンが4～5日ほど経つと強制終了され、再起動していました。

原因を探す

最初は、再起動のルールがあるのだと思っていました。他チームのバックエンドサーバーは、実際に週に一度マシンを再起動していると聞きました。しかし、インスタンスがメモリを20GB も使うのは、再起動のルールとは関係なく、何かがおかしい状態でした。



インスタンス1台のメモリ、5日で1GB から32GB まで

数日間モニタリングした結果、API サーバーのどこかでメモリリークが起きてインスタンスのメモリを使い切り、最後に OOM で強制終了されていることがわかりました。ところが Sentry にも AWS のモニタリングツールにも原因になりそうな痕跡が残っておらず、デバッグは難航しました。

メモリダンプを取る

チームリーダーから、本番サーバーのメモリダンプを取って調べようというアイデアをもらいました。正直、気が進みませんでした。ダンプを取っている間にサービスに障害が出るかもしれない、ダンプを持っても読み方がわからないので、意味のある結果を見つけるのは難しそうでした。それでも他に方法がなく、ひとまずやってみることにしました。

調べてみると、メモリダンプのツールがいくつかありました。チームリーダーに共有し、本番サーバーに直接アクセスして、2つのツールでメモリをスキャンしました。1つ目のツールでは何も得られませんでした。2つ目のツールで、同じエラーが何度も発生したと思われるダンプのログを見つけました。

原因と解決

原因は、アプリのプッシュ送信を並列化したコードにありました。並列処理のためにスレッドを作るのですが、スレッドの中で例外が発生すると、そのスレッドが回収されずにそのまま残り、メモリリークが積み重なっていました。スレッド内部の処理ロジックに try/catch を付けてデプロイすると、それ以降すべてのインスタンスが1GB ほどしか使わない正常な状態に戻りました。

最初は解決策がまったく思い浮かばず、ストレスを感じていました。結局、自分で開いて中をのぞいてみれば、答えは思ったより近くにあるということを学んだ作業でした。