

Webサービスの初期開発とマイクロサービス・BFF

Microservices Meetup vol.5 2017/3/30

Akatsuki Inc. 高松 真平

私は誰?

- 高松 真平 (@shimpeiws)
- Webエンジニア @ アカツキ LX事業部
 - ・ **Rails**と**JavaScript**を半々くらい書いている



■ LX(ライブエクスペリエンス)事業部

- ・ 世界中の人々にライブ（生の、リアルな、ワクワクする）
- ・ エクスペリエンス（体験）を提供する事業です。



1. 要件とアーキテクチャ

2. 初期開発での選択と集中と負債

3. BFFのプロトタイプ実装



1. 要件とアーキテクチャ

新規開発中のサービスの話

1. 要件とアーキテクチャ

■ 機能要件

- **Web**からスタートする
 - 後からネイティブアプリが出る可能性は高い
- toC部分はスマートフォンユーザが多数と予想される
- **ユーザインタラクションが多いUI**
- **SEO**に強い早くて構造化されたマークアップ
 - 検索からの流入の比重が大きい
- 長期的な運用になる見込み
 - **高い保守性と(将来的な)スケーラビリティが必要**

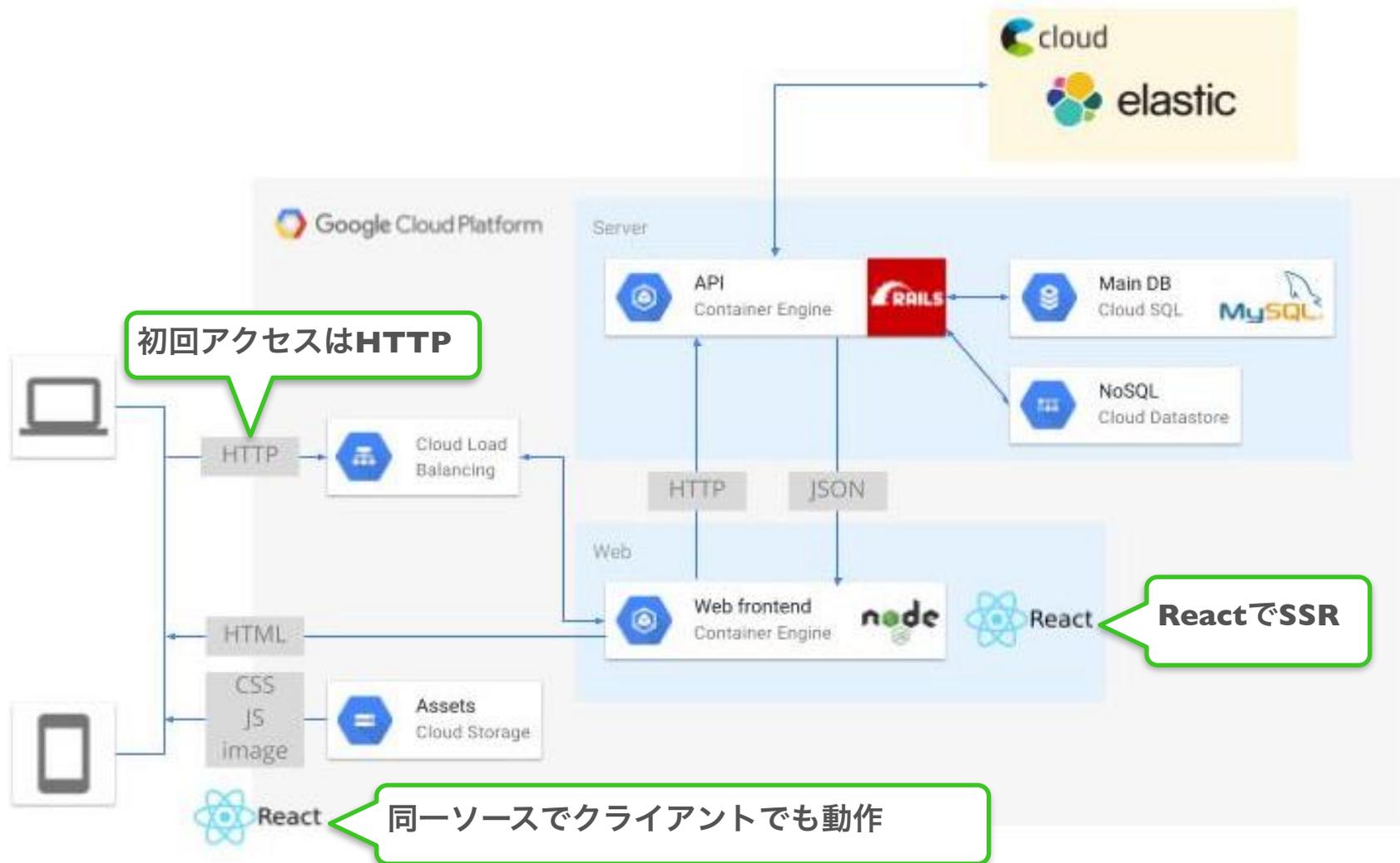
1. 要件とアーキテクチャ

■ アーキテクチャ

- Webフロントエンド
 - **Single Page Application**として構築
 - React + Redux
 - **Isomorphic JavaScript** の構成で、サーバサイドレンダリング
- サーバサイド
 - **Rails**
 - MySQL
- インフラ
 - Google Cloud Platform (Google Container Engine)
 - **Kubernetes**でコンテナ運用

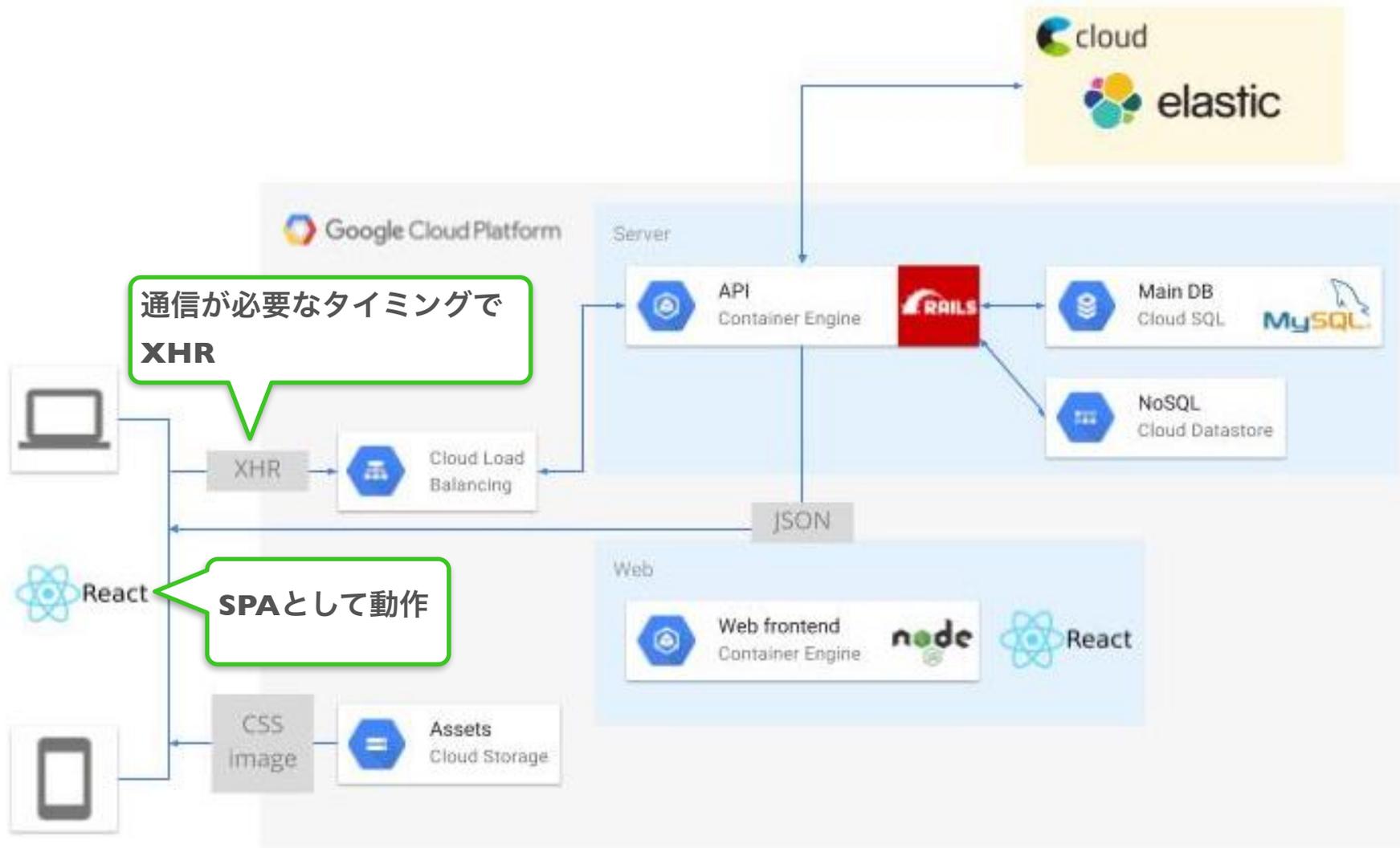
1. 要件とアーキテクチャ

- Web、Serverの2サービスで稼働している



1. 要件とアーキテクチャ

- Web、Serverの2サービスで稼働している





2. 開発初期での選択と集中と課題

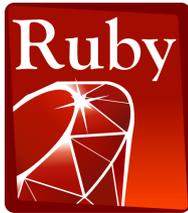
2. 開発初期での選択と集中と課題

■ ここまでの開発のチーム・開発内容・ハイライト

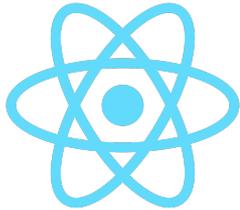
	2016/9~2016/11 (立ち上げ期)	2016/12~2017/3 (開発初期)	2017/4~ (開発中期)
チーム	<ul style="list-style-type: none"> - 1チーム - サーバ + Web 4名 	<ul style="list-style-type: none"> - 1チーム - サーバ + Web 5名 	<ul style="list-style-type: none"> - ミッション毎に4チームに分割 - ユーザ系、サービス基盤系、インフラ系...
開発内容	<ul style="list-style-type: none"> - 開発フロー整備 - QA・Production環境構築 - 基本機能を一通り 	<ul style="list-style-type: none"> - アイテムタイプの追加 - 口コミの実装 - 全文検索(Elasticsearch) - バグ・負債を若干返済 	<ul style="list-style-type: none"> - グロースハック施策 - サービスの基盤となる機能の開発 - インフラ・ミドルウェア・デプロイの整備
ハイライト	<ul style="list-style-type: none"> - アーキテクチャ・設計指針決定 - 開発ボリューム抑えきれず混乱 	<ul style="list-style-type: none"> - アーキテクチャに大きく手は入っていない - スクラム開発の運用を手探り 	<ul style="list-style-type: none"> - チームの分割 - マイクロサービス化の推進 - 中長期的な運用の開始

2. 開発初期での選択と集中と課題

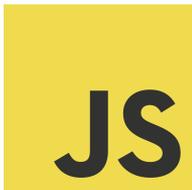
- ここまでの開発の分量 (テストコードは除く)



15,000行(モデル数 100弱)



60,000行

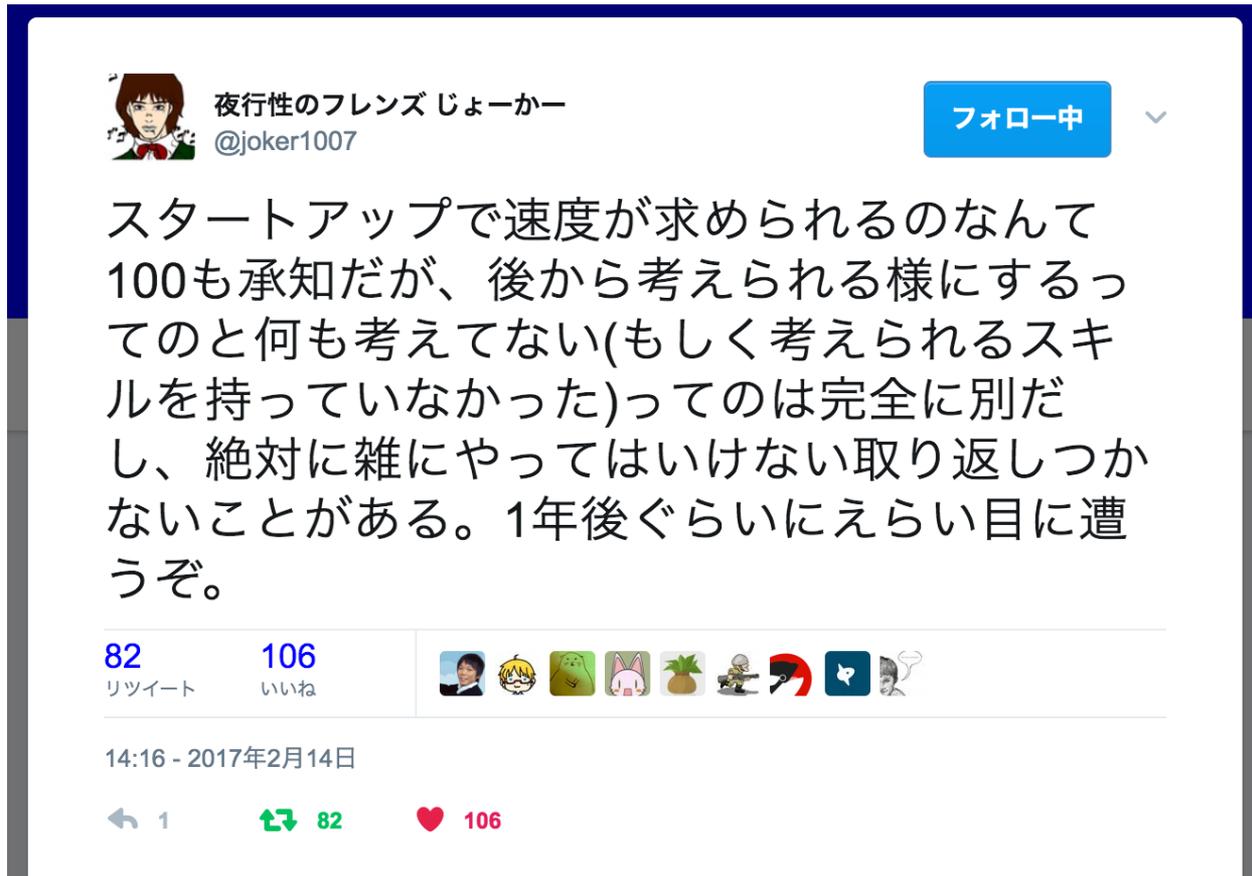


25,000行

結構なスピードで開発を進めて来た

2. 開発初期での選択と集中と課題

- 立ち上げ期～開発初期にアーキテクチャ・設計上考えたいこと



The image shows a screenshot of a tweet from the user '夜行性のフレンズ じょーかー' (@joker1007). The tweet text discusses the challenges of starting a project with speed requirements, noting that it's often easier to start than to maintain quality over time. The tweet has 82 retweets and 106 likes. The interface includes a 'フォロー中' (Following) button and a list of users who interacted with the tweet.

夜行性のフレンズ じょーかー
@joker1007

フォロー中

スタートアップで速度が求められるのなんて100も承知だが、後から考えられる様にするってのと何も考えてない(もしくは考えられるスキルを持っていなかった)ってのは完全に別だし、絶対に雑にやってはいけない取り返すつかないことがある。1年後ぐらいにえらい目に遭うぞ。

82 リツイート 106 いいね

14:16 - 2017年2月14日

1 82 106

<https://twitter.com/joker1007/status/831371523818614789>

2. 開発初期での選択と集中と課題

■ 立ち上げ期～開発初期の**選択と集中**

- ・ 全面的に**Single Page Application + SSR**の構成にする
- ・ **Why**
 - ・ 複雑なユーザインタラクションを実現したい
 - ・ イニシャルビューの表示を早くしたい
- ・ **What**
 - ・ **Single Page Application + SSR**の実現
- ・ **How**
 - ・ **React + Redux (on Isomorphic Javascript)** の採用

2. 開発初期での選択と集中と課題

■ 立ち上げ期～開発初期の**選択と集中**

- ・ 全面的に**Single Page Application + SSR**の構成にする
- ・ **Pros**
 - ・ ダイナミックなWebフロントエンドを作りやすい
 - ・ SPAの構成で一本化できる(ルーティング、State管理...)
- ・ **Cons**
 - ・ **開発工数**がかかる (例えばRails(ERBなど) + jQueryに比べて)
 - ・ Node.jsのフロントサーバが一台増え、**デプロイ・管理が複雑化**した

2. 開発初期での選択と集中と課題

■ 立ち上げ期～開発初期の選択と集中

- ・ **API設計**の単純化
- ・ **Why**
 - ・ チーム開発だが、ドメイン設計にかける**時間を十分にとれない**
 - ・ 画面にべったりそったAPIで**保守性**を下げたくない
- ・ **What**
 - ・ かなり**細かな粒度でのリソース指向設計**で統一する
 - ・ (基本的に)**RESTful API**
- ・ **How**
 - ・ 1モデル = 1APIになることが多い、重要な(大きい)ものだけリソース単位を設計
 - ・ 初期の**レビューでチーム内の意識統一**をしていった

2. 開発初期での選択と集中と課題

■ 立ち上げ期～開発初期の**選択と集中**

- ・ **API設計**の単純化

- ・ **Pros**

- ・ 画面にべったりそったAPIは避けられた
- ・ 設計・実装が**楽でバグも少なかった**

- ・ **Cons**

- ・ **クライアントサイドの実装に負荷**がよった

=> このあとの ”課題” に基づく

2. 開発初期での選択と集中と課題

■ 立ち上げ期～開発初期の選択と集中の結果残った課題

- ・クライアントサイドの **リクエスト・ハンドリングが複雑化**する

```
JS gotanda.js x
1  export function initActivities(activityId) {
2    return async(dispatch) => {
3      const activityData = await Promise.all([
4        activitiesShow(activityId),
5        activityAppealsIndex({ activityId }),
6        activityStrengthsShow(activityId),
7        calendarIndex(activityId),
8        courseTimesShow(activityId, moment()),
9        pricesIndex(activityId)
10     ])
11     const activity = activityData[0]
12     dispatch(setActivity(activity, activityData[1], activityData[2]))
13     dispatch(setCalendar(activityData[3]))
14     dispatch(setCourseTime(activityData[4]))
15     dispatch(setPrices(activityData[5]))
16
17     const company = await companyShow(activity.companyId)
18     dispatch(setCompany(company))
19
20     const area = await areaShow(activity.areaId)
21     dispatch(setArea(area))
22   }
23 }
24
```

6並列

直列

直列

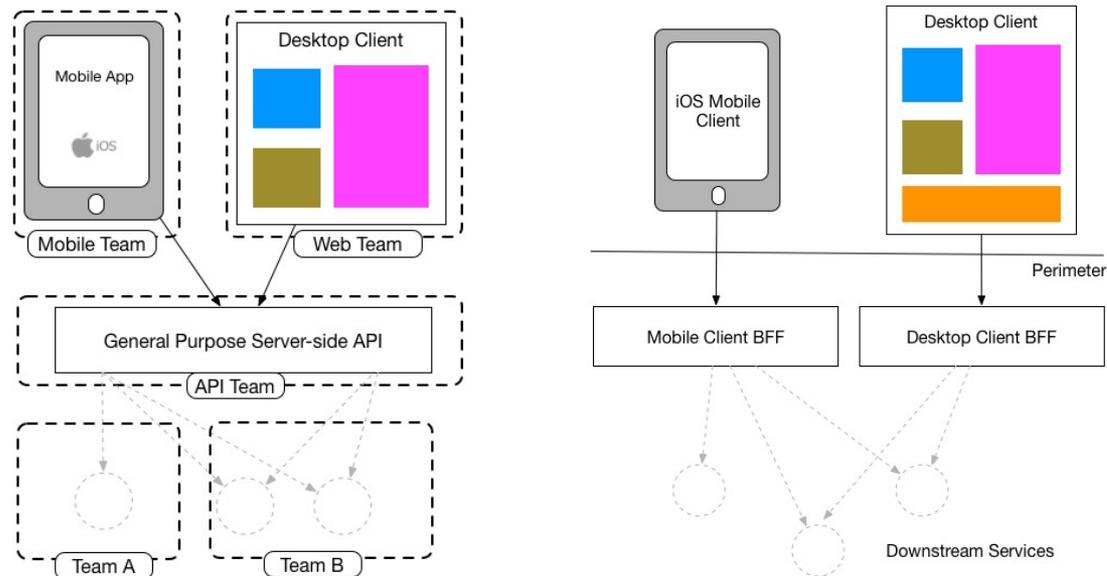
※ある詳細画面の実装イメージ

アーキテクチャ上の解決余地はのこっている
そう、**BFF**ならね!!!

2. 開発初期での選択と集中と課題

■ BFF(Backend For Frontend)

- ・元Sound CloudのPhil Calçadoが提唱
- ・クライアントの種別毎に中間レイヤを作る



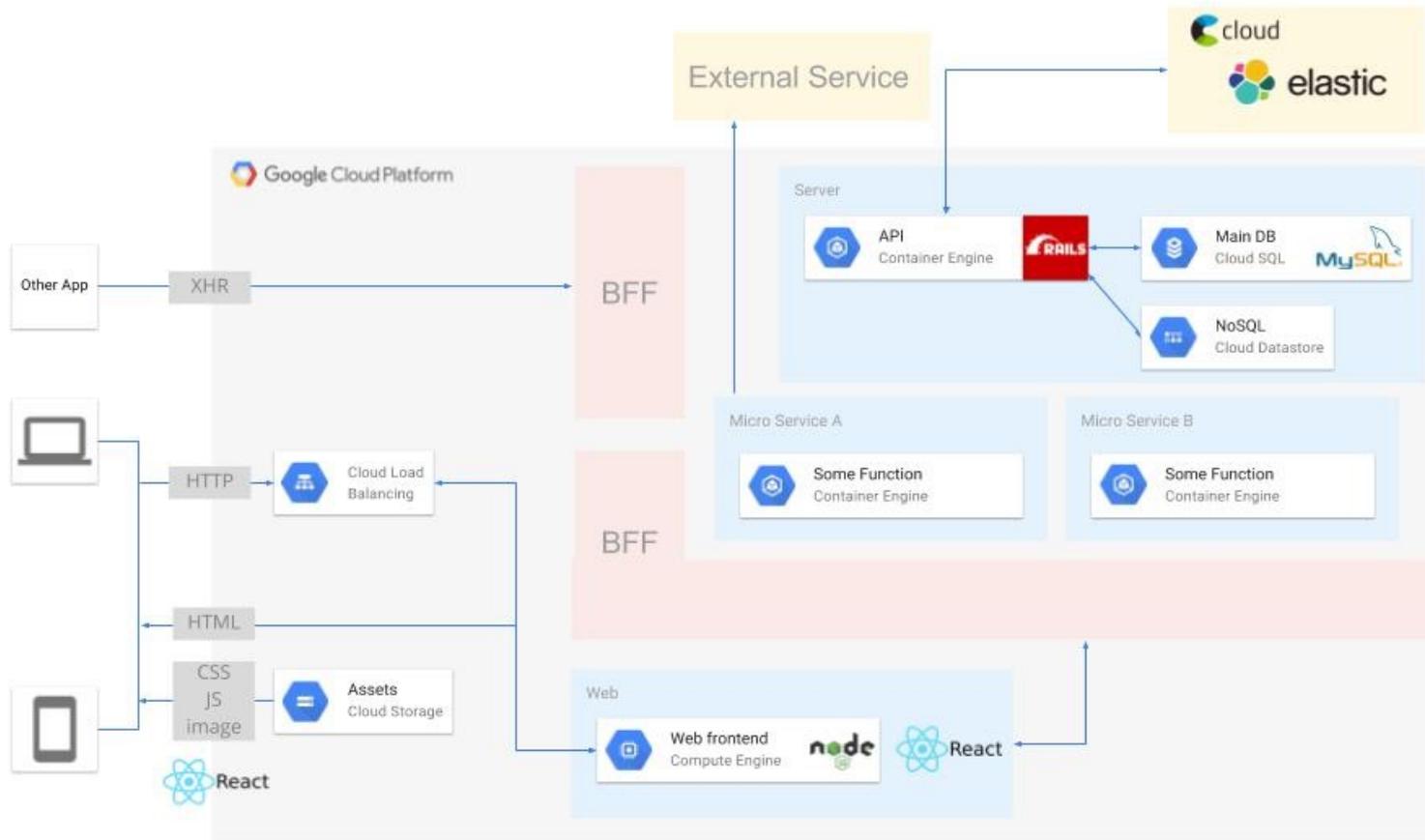
“Pattern: Backends For Frontends” Sam Newman

<http://samnewman.io/patterns/architectural/bff/>

2. 開発初期での選択と集中と課題

■ BFF導入後のアーキテクチャ

- ・ 2017年中にはこんな感じになる???



2. 開発初期での選択と集中と課題

- 結局開発初期で何を選択して、今後に何を残したのか?
 - ・ 開発初期
 - ・ 立ち上げ期にも ”オーケストレーション層を作るか?” などの話は出ていた
 - ・ が、**決めきれなかった & 実装する余力がなかった**
 - ・ 今後のアーキテクチャ・開発にどんな意味があったのか???
 - ・ **後から考えられる余地**を残したかった
 - ・ **レイヤーを1枚足す考え方**はこういう場合に都合が良い
 - ・ **API Gateway・BFF**はAPIに関する “レイヤーを足す” アーキテクチャ
 - ・ そのためにバックエンドとフロントエンドが密結合しているのは都合が悪い

要するに

API関連のアーキテクチャ決定をペンディングした(できた)



3. BFFのプロトタイプ実装

3. BFFのプロトタイプ実装

■ BFFに求められる要件 (Why)

- ・ クライアントから見て
 - ・ 画面表示に必要なデータを**まとめて取得したい**
 - ・ でも、**レスポンスが遅くなるのはNG**
- ・ 設計・アーキテクチャの観点から
 - ・ ゲートウェイとしてバックエンドの前に置かれる
 - ・ **遅いと全体のレスポンス悪化**を招く
 - ・ **スケーラビリティ**が要求される
 - ・ フロントエンドと密結合するので**更新頻度は高くなる**
 - ・ フロントエンドを触るエンジニアが気軽に変更できる必要がある
 - ・ マイクロサービスが抱える分散システムの難しさは当然ある
 - ・ 分散トレースや診断・修復のシステム化も考えていく必要性

3. BFFのプロトタイプ実装

■ BFFの設計 (What・How)

・ What

- ・ 要は**APIクライアント**を作る

・ How

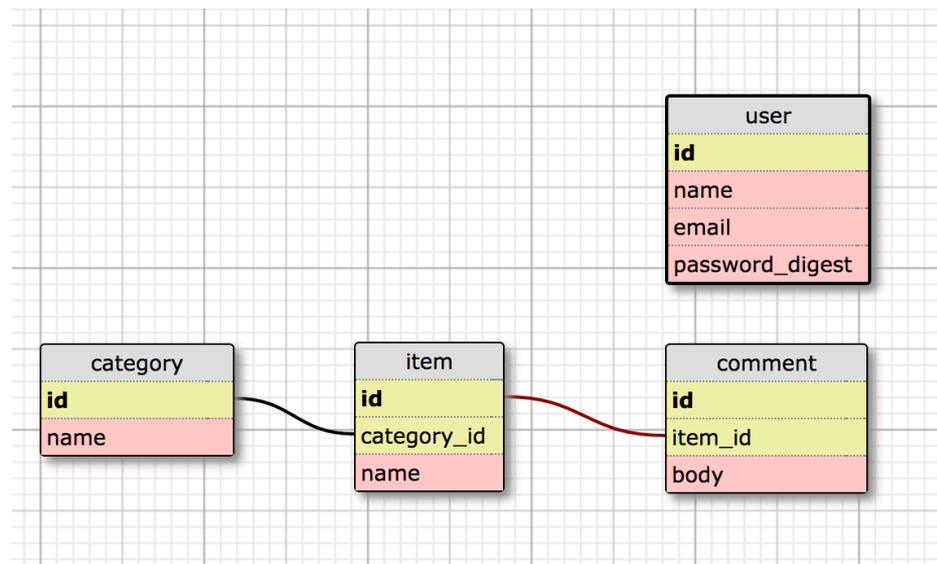
・ **言語・フレームワーク**の選定

- ・ スケーラビリティ・速度に関して優位性があること
- ・ フロントエンドを触るエンジニアも学習が用意であること
 - ・ 自分たちのチームだと選択肢は Go or Elixirか...

3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [1: 仕様とER図]

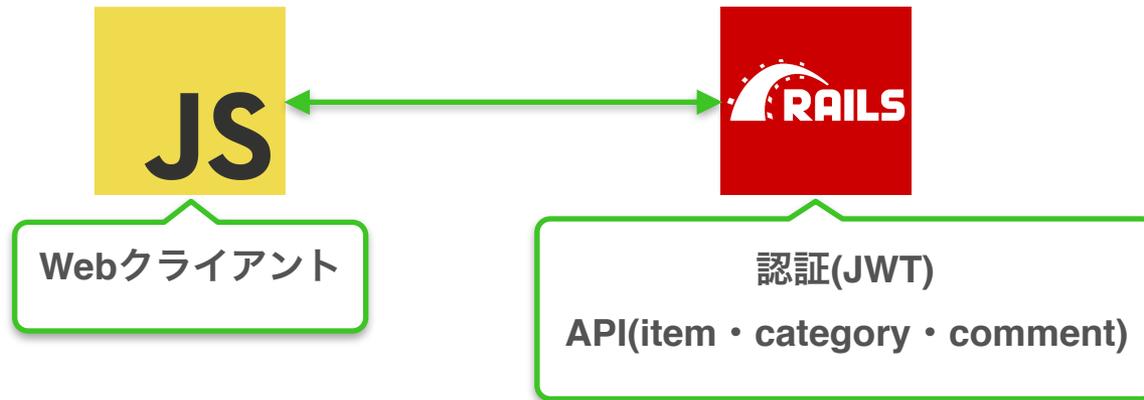
- ・ こんな仕様を想定してみる
 - ・ 一覧に商品と商品についてのコメントを表示する
 - ・ 商品にはカテゴリが設定されている
 - ・ 会員専用サービスなのでログイン済みユーザーのみ閲覧可能
 - ・ APIの全てのエンドポイントにユーザ認証がある、認証はJWTで行う



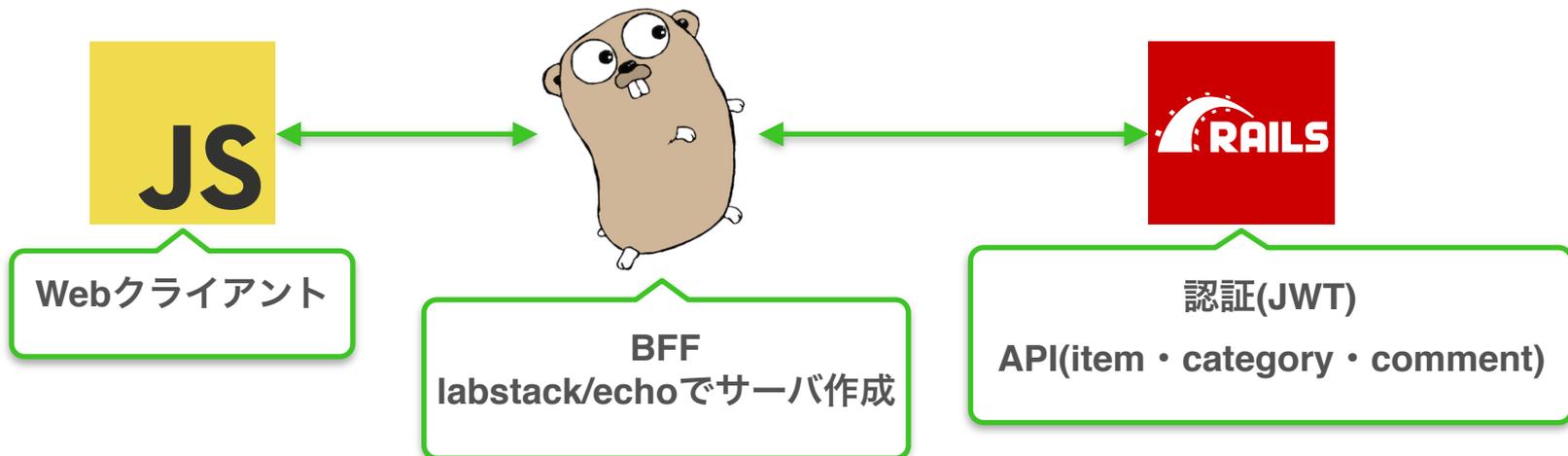
3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [2: 構成図]

・ BEFORE



・ AFTER



3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [3: BEFORE] ※Web Client(JS)のコード

```
80 export const getItem = createAction('GET_ITEMS', async () => {
81   const result = await getItemAndCategories()
82   const comments = await getComments(result.items)
83   console.info('result', result)
84   console.info('comments', comments)
85   return Object.assign({}, result, { comments })
86 })
```

メインの取得処理
- itemとcategory
- comment

```
52 const getItemAndCategories = async () => {
53   const instance = await getInstance()
54   return await axios
55     .all([
56       instance.get('items'),
57       instance.get('categories')
58     ])
59     .then(() => {
60       axios.spread((itemResult, categoryResult) => {
61         return {
62           items: itemResult.data.data,
63           categories: categoryResult.data.data
64         }
65       })
66     })
67 }
```

itemとcategory取得処理
並列で取得

comment取得処理
itemのcount分を並列で取得

```
69 const getComments = async (items) => {
70   const instance = await getInstance()
71   return await axios
72     .all(
73       items.map((item) => instance.get(`comments/${item.id}`))
74     )
75     .then((comments) => {
76       return comments.map((comment) => comment.data.data)
77     })
78 }
```

3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [3: BEFORE] ※Web Client(JS)のコード

```
27  const getJwtToken = async () => {
28    axios.defaults.baseURL = 'http://localhost:3000/'
29    axios.defaults.timeout = 3000
30    axios.defaults.headers.post['Content-Type'] = 'application/json'
31    const userTokenResult = await axios.post(
32      'user_token',
33      {
34        auth: {
35          email: 'test@example.com',
36          password: 'test123'
37        }
38      }
39    )
40    return userTokenResult.data.jwt
41  }
42
43  const getInstane = async () => {
44    const jwtToken = await getJwtToken()
45    return axios.create({
46      headers: {
47        'Authorization': `Bearer ${jwtToken}`
48      }
49    })
50  }
```

JWTトークンの取得
/user_token へのPOST

APIクライアントの取得
ヘッダにJWTトークンが入る

3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [4: AFTER] ※ BFF(Go)のコード

```
191 e.GET("/items", func(c echo.Context) error {
192     token, err := getJwtToken()
193     if err != nil {
194         return c.String(http.StatusForbidden, err.Error())
195     }
196     fmt.Println("token", token)
197
198     // TODO: Do concurrently
199     items, getItemErr := getItems(token)
200     if getItemErr != nil {
201         return c.String(http.StatusNotFound, getItemErr.Error())
202     }
203     fmt.Println("items", items)
204
205     categories, getCategoryErr := getCategories(token)
206     if getCategoryErr != nil {
207         return c.String(http.StatusNotFound, getCategoryErr.Error())
208     }
209     fmt.Println("categories", categories)
210
211     // TODO: implement
212     itemIds := []int{1}
213     comments, getCommentErr := getComments(token, itemIds)
214     if getCommentErr != nil {
215         return c.String(http.StatusNotFound, getCommentErr.Error())
216     }
217     fmt.Println("comments", comments)
218
219     response := ItemsResponse{Items: *items, Categories: *categories, Comments: *comments}
220     // response := ItemsResponse{}
221     jsonBytes, err := json.Marshal(response)
222
223     fmt.Println("response json", string(jsonBytes))
224     return c.String(http.StatusOK, string(jsonBytes))
225 })
```

labstack/echo でサーバを立てている

メインの取得処理

- JWTのトークン取得
- itemとcategory
- comment

3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [4: AFTER] ※ BFF(Go)のコード

```
62 func getJwtToken() (string, error) {
63     requestStr := `{"auth":{"email":"test@example.com","password":"test123"}}`
64     req, err := http.NewRequest(
65         "POST",
66         "http://localhost:3000/user_token",
67         bytes.NewBuffer([]byte(requestStr)),
68     )
69
70     if err != nil {
71         return "", err
72     }
73
74     req.Header.Set("Content-Type", "application/json")
75
76     client := &http.Client{}
77     resp, err := client.Do(req)
78
79     if err != nil {
80         return "", err
81     }
82     defer resp.Body.Close()
83
84     body, err := ioutil.ReadAll(resp.Body)
85     if err != nil {
86         return "", err
87     }
88
89     jwtResponse := new(JwtToken)
90     fmt.Println(string(body))
91     jsonParseError := json.Unmarshal(body, &jwtResponse)
92     if jsonParseError != nil {
93         return "", jsonParseError
94     }
95
96     return jwtResponse.Jwt, nil
97 }
```

JWTトークンの取得
/user_token へのPOST

```
99 func getRequest(path, token string) ([]byte, error) {
100     req, err := http.NewRequest(
101         "GET",
102         "http://localhost:3000/" + path,
103         nil,
104     )
105     if err != nil {
106         return nil, err
107     }
108
109     req.Header.Add("Authorization", string("Bearer " + token))
110     req.Header.Set("Content-Type", "application/json")
111
112     client := &http.Client{}
113     resp, err := client.Do(req)
114     defer resp.Body.Close()
115     if err != nil {
116         return nil, err
117     }
118
119     body, err := ioutil.ReadAll(r
120     if err != nil {
121         return nil, err
122     }
123
124     return body, nil
125 }
```

GETリクエストのラッ
パー
JWTトークンをヘッダー
にセットしている

3. BFFのプロトタイプ実装

■ BFFのプロトタイプ実装 [4: AFTER] ※ BFF(Go)のコード

```
127 func getItems(token string) (*ItemList, error) {
128     > body, err := getRequest("items", token)
129     > if err != nil {
130     >     > return nil, err
131     > }
132     >
133     > itemResponse := new(ItemList)
134     > fmt.Println("getItemsResponse", string(body))
135     > jsonParseError := json.Unmarshal(body, &itemResponse)
136     > if jsonParseError != nil {
137     >     > return nil, jsonParseError
138     > }
139     >
140     > return itemResponse, nil
141 }
```

```
17 type ItemList struct {
18     > Data []Item
19 }
20
21 type Item struct {
22     > Id string
23     > Attributes ItemDetail
24 }
25
26 type ItemDetail struct {
27     > Name string
28 }
```

ItemのAPIリクエスト部分
レスポンスと同じStructを作っている

3. BFFのプロトタイプ実装

- BFFのプロトタイプ実装 [4: AFTER] ※ BFF(Go)のコード

```
219 >> response := ItemsResponse{Items: *items, Categories: *categories, Comments: *comments}
220 >> // response := ItemsResponse{}
221 >> jsonBytes, err := json.Marshal(response)
222 >>
223 >> fmt.Println("response json", string(jsonBytes))
224 >> return c.String(http.StatusOK, string(jsonBytes))
225 >> })
226
```

```
56 type ItemsResponse struct {
57     Items ItemList
58     Categories CategoryList
59     Comments CommentList
60 }
```

最終的にレスポンスを返す部分
レスポンスの形のStructをtoJSONする

3. BFFのプロトタイプ実装

- BFFのプロトタイプ実装 [4: AFTER] ※Web Client(JS)のコード

```
92 export const GetItemsBff = createAction('GET_ITEMS', async () => {  
93   · const result = await axios.get('http://localhost:1323/items')  
94   · console.info('result', result)  
95   · return result  
96 })
```

BFFのエンドポイントをコール
APIは1本だけ!!!

3. BFFのプロトタイプ実装

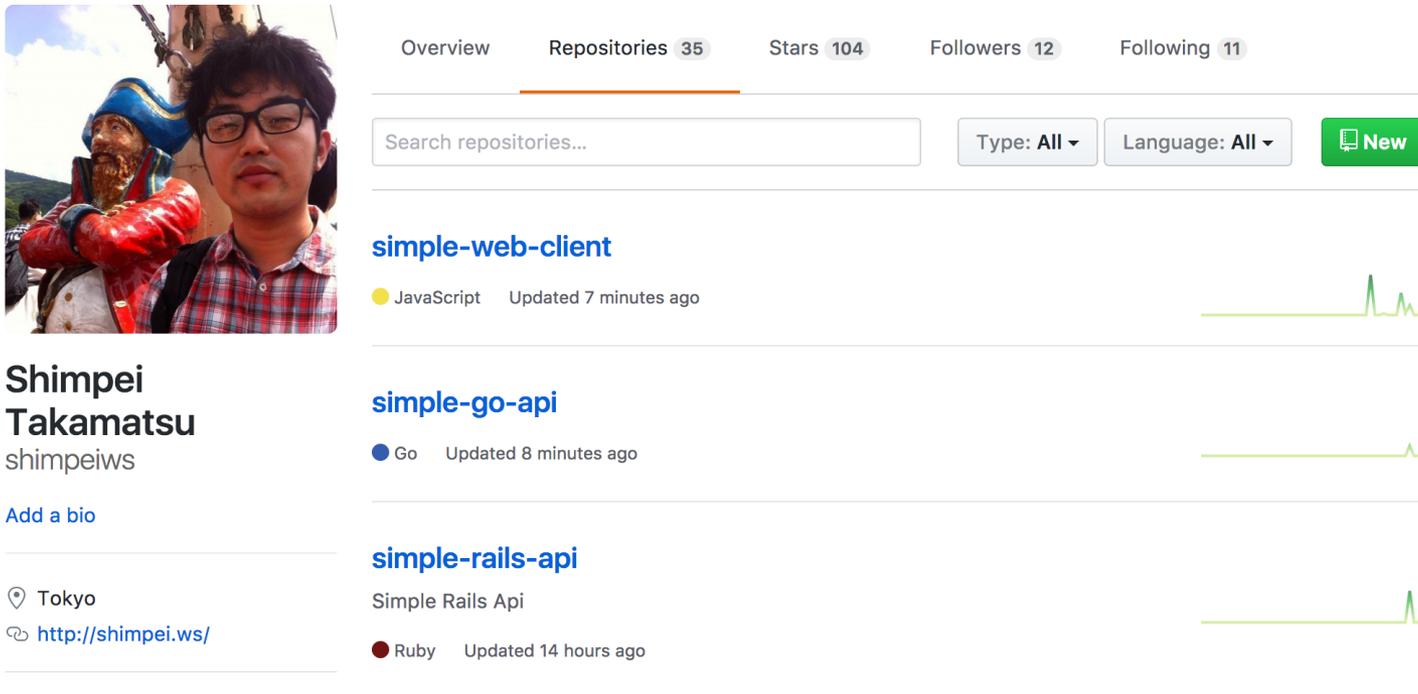
■ BFFのプロトタイプ実装 [5: どうか?]

- ・ フロントエンドからBFFに**APIリクエストのハンドリングが移った**
- ・ フロントエンドから見ると**画面に対応するAPI**を1本投げるだけで良い
- ・ BFFの実装は単純だけど**ちょっと煩雑**
 - ・ チームのメンバーに”めんどくさい”と言われそう...
 - ・ マッピング部分はgeneratorを用意するなり、工夫必要そう

3. BFFのプロトタイプ実装

■ 今回のサンプルコード

- Webクライアント: **shimpeiws/simple-web-client**
- BFF(Go): **shimpeiws/simple-go-api**
- Rails API: **shimpeiws/simple-rails-api**



The screenshot shows the GitHub profile of Shimpei Takamatsu (shimpeiws). The profile includes a profile picture, a bio, location (Tokyo), and website (http://shimpei.ws). The repository list shows three repositories: simple-web-client (JavaScript, updated 7 minutes ago), simple-go-api (Go, updated 8 minutes ago), and simple-rails-api (Ruby, updated 14 hours ago). The repository statistics show 35 repositories, 104 stars, 12 followers, and 11 following.

Overview **Repositories 35** Stars **104** Followers **12** Following **11**

Search repositories... Type: All Language: All **New**

simple-web-client
JavaScript Updated 7 minutes ago

simple-go-api
Go Updated 8 minutes ago

simple-rails-api
Simple Rails Api
Ruby Updated 14 hours ago

Shimpei Takamatsu
shimpeiws

Add a bio

📍 Tokyo
🌐 <http://shimpei.ws>

まとめ

■ まとめ

・ 開発のこれまで

- ・ 開発スピードを上げて初期開発を乗り切ってきた
- ・ **アーキテクチャが決めきれず、API関連の設計を単純化**しすることにした
- ・ 結果、**フロントエンドに重さが寄った**

・ 開発のこれから

- ・ **BFF**層を導入していきたい
 - ・ フロントエンドのハンドリング複雑化の回避
 - ・ マイクロサービス化からフロントエンドを守る
- ・ **後から考える余地のあるアーキテクチャ・設計、大事**

We are hiring!!!

中途採用 > 日本 > Web エンジニア

Web エンジニア

Kubernetesでマイクロサービスを追求するエンジニア募集!!!



株式会社アカツキ

埋め込む

いいね! 0

ツイート

B! Bookmark 0

