Akatsuki

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。

目次

Akatsuki

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

Akatsuki

- **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

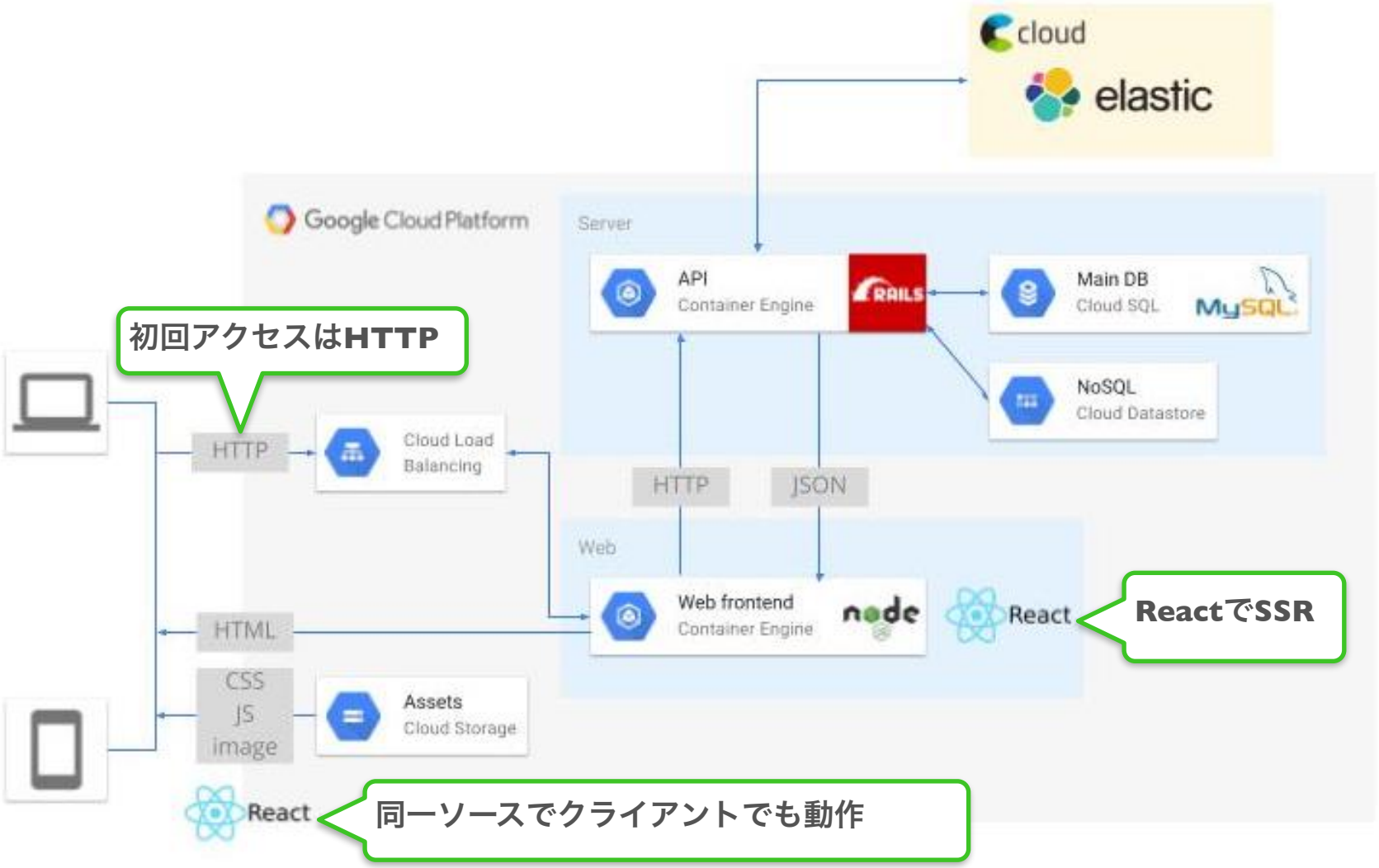
Akatsuki

- **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

Akatsuki

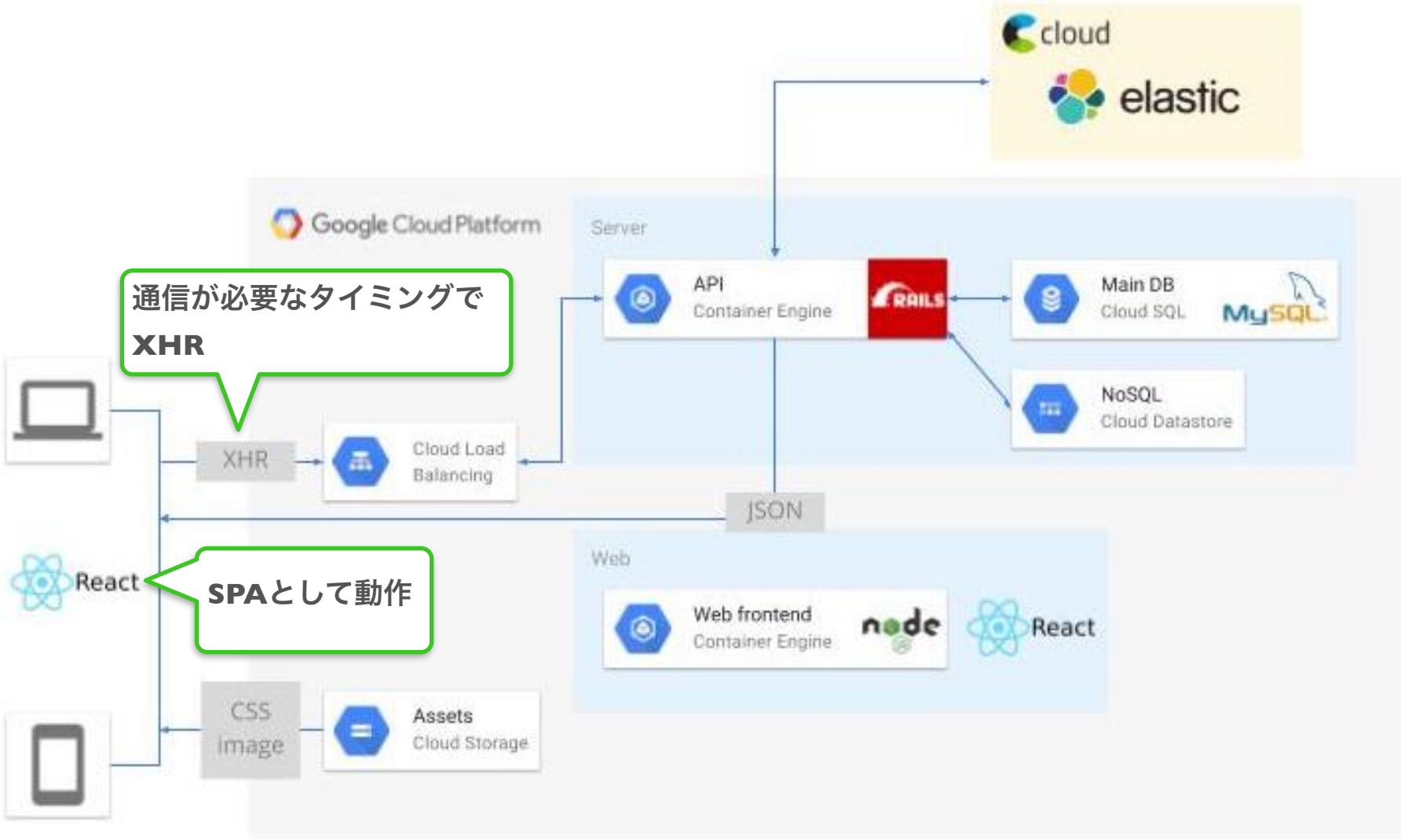
- Web、Serverの2サービスで稼働している



1. 要件とアーキテクチャ

Akatsuki

- Web、Serverの2サービスで稼働している



Akatsuki

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

Akatsuki

- ここまでの開発のチーム・開発内容・ハイライト

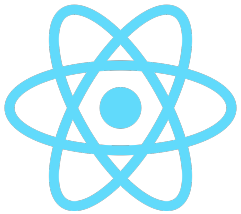
	2016/9~2016/11 (立ち上げ期)	2016/12~2017/3 (開発初期)	2017/4~ (開発中期)
チーム	- 1チーム - サーバ＋Web 4名 	- 1チーム - サーバ＋Web 5名 	- ミッション毎に4チームに分割 - ユーザ系、サービス基盤系、インフラ系…
開発内容	- **開発フロー**整備 - QA・Production環境構築 - **基本機能**を一通り	- アイテムタイプの追加 - 口コミの実装 - 全文検索(Elasticsearch) - **バグ・負債**を若干返済	- **グロースハック**施策 - サービスの基盤となる機能の開発 - **インフラ・ミドルウェア・デプロイ**の整備
ハイライト	- **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	- **アーキテクチャに大きく手は入れていない** - **スクラム開発**の運用を手探り	- **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

Akatsuki

- 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

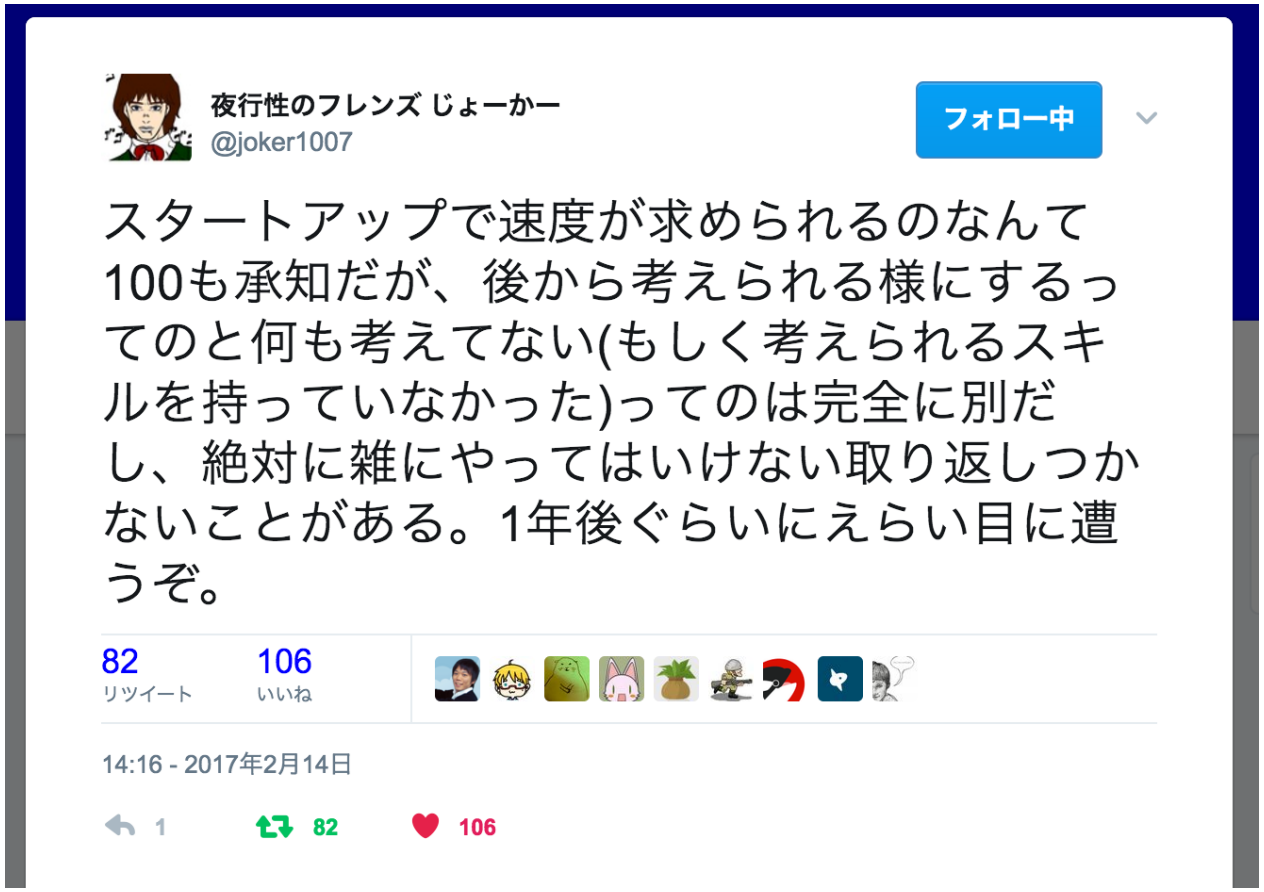
25,000行

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

Akatsuki

- 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



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

2. 開発初期での選択と集中と課題

Akatsuki

- 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

Akatsuki

- 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

Akatsuki

- **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

Akatsuki

- **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” につづく

2. 開発初期での選択と集中と課題

Akatsuki

- 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

※ある詳細画面の実装イメージ

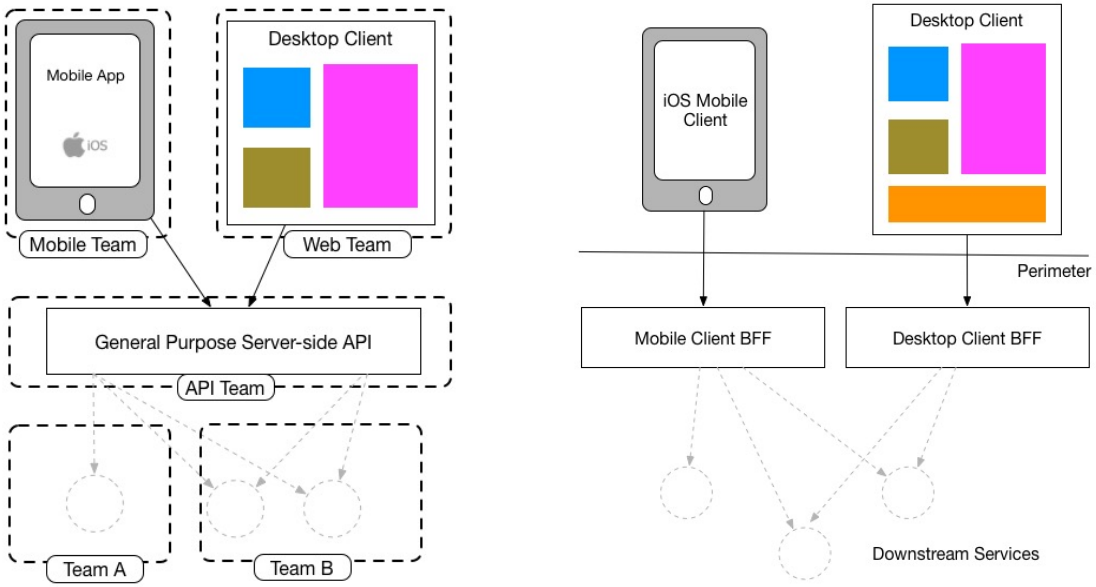
アーキテクチャ上の解決余地はのこっている

そう、BFFならね!!!

2. 開発初期での選択と集中と課題

Akatsuki

- **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



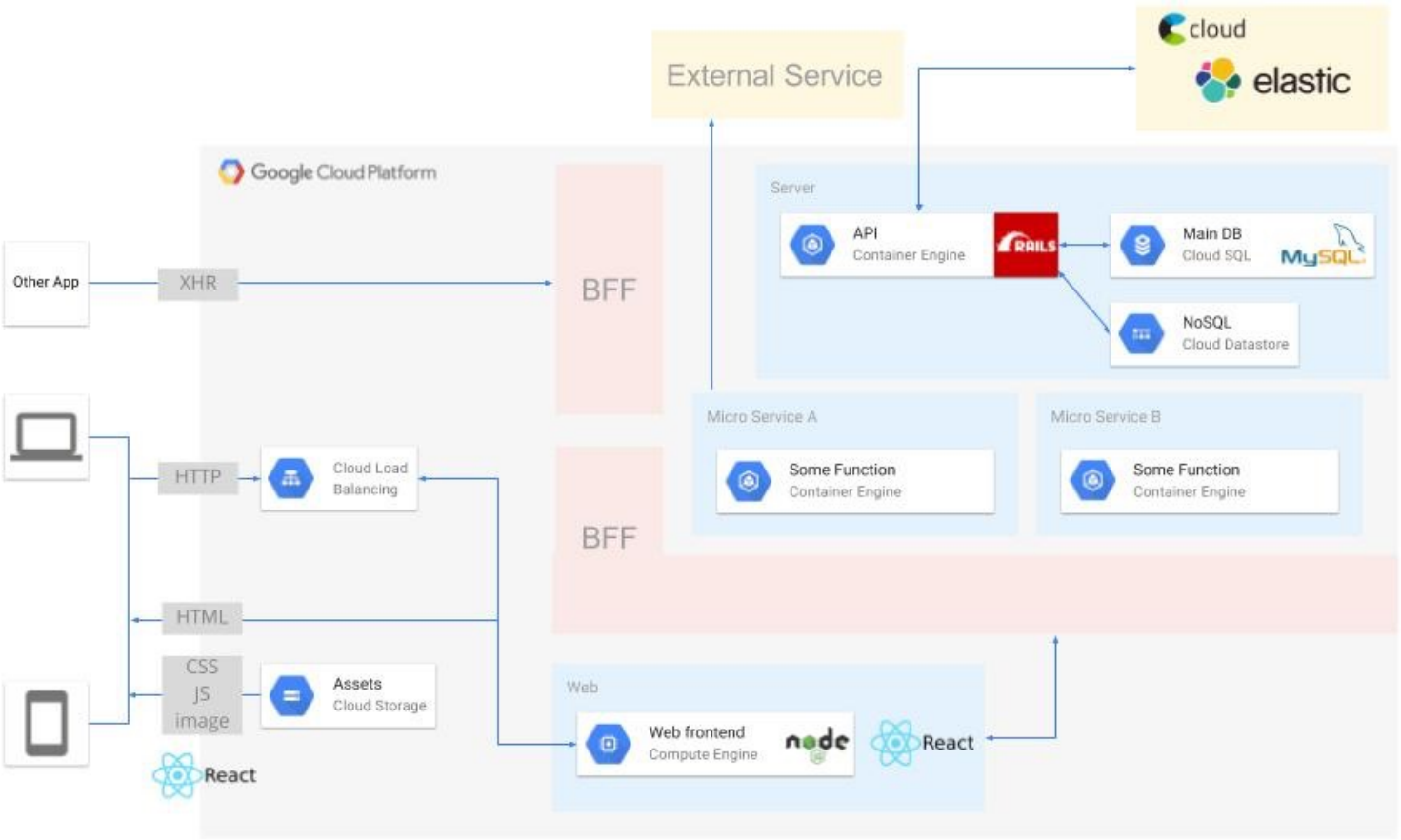
“Pattern: Backends For Frontends” Sam Newman

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

2. 開発初期での選択と集中と課題

Akatsuki

- **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

Akatsuki

- **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に関しての “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)

Akatsuki

3. BFFのプロトタイプ実装

3. BFFのプロトタイプ実装

Akatsuki

- **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

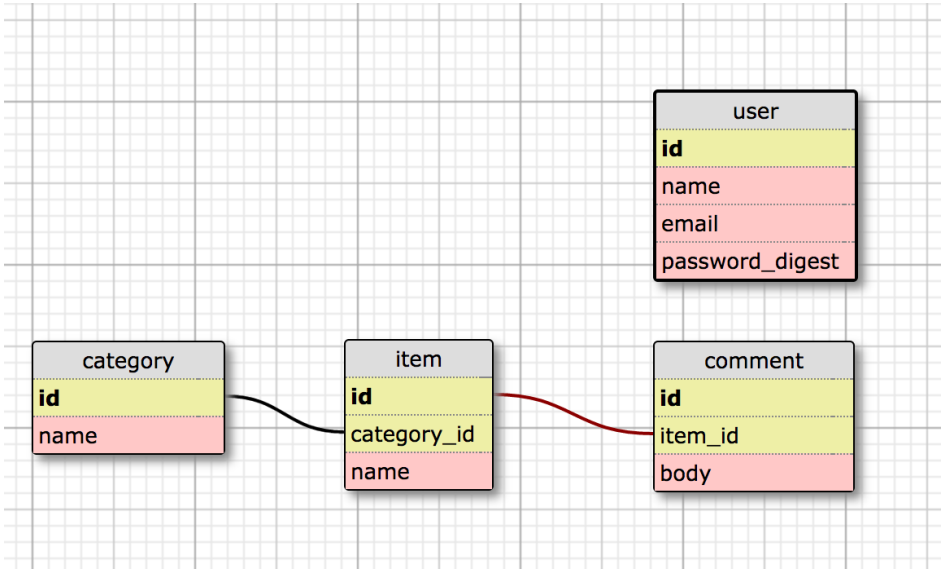
Akatsuki

- **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

Akatsuki

- **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**商品と商品についたコメントを表示**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

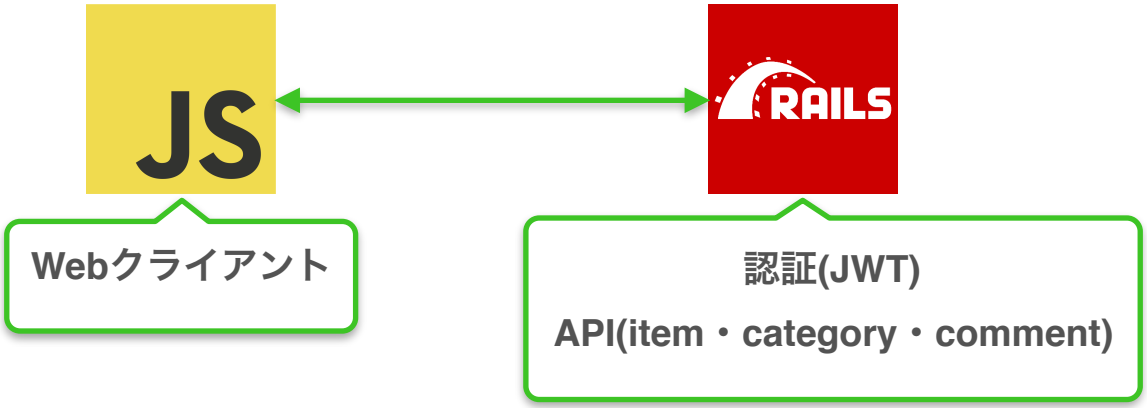


3. BFFのプロトタイプ実装

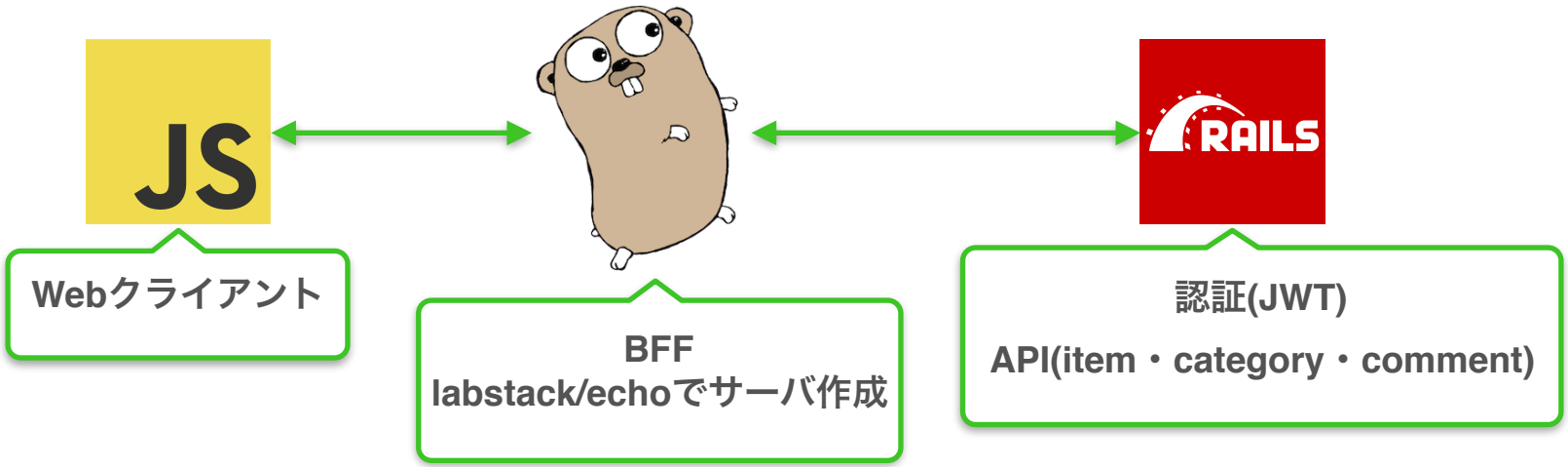
Akatsuki

- 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

Akatsuki

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

```
80  export const getItems = createAction('GET_ITEMS', async () => {
81    const result = await getItemsAndCategories()
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 getItemsAndCategories = async () => {
53    const instance = await getInstane()
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 getInstane()
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

Akatsuki

- 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

Akatsuki

- 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

Akatsuki

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

```go
func getJwtToken() (string, error) {
    requestStr := `{"auth":{"email":"test@example.com","password":"test123"}}`
    req, err := http.NewRequest(
        "POST",
        "http://localhost:3000/user_token",
        bytes.NewBuffer([]byte(requestStr)),
    )

    if err != nil {
        return "", err
    }

    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{}
    resp, err := client.Do(req)

    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    jwtResponse := new(JwtToken)
    fmt.Println(string(body))
    jsonParseError := json.Unmarshal(body, &jwtResponse)
    if jsonParseError != nil {
        return "", jsonParseError
    }

    return jwtResponse.Jwt, nil
}
```

JWTトークンの取得
/user_token へのPOST

```go
func getRequest(path, token string) ([]byte, error) {
    req, err := http.NewRequest(
        "GET",
        "http://localhost:3000/" + path,
        nil,
    )
    if err != nil {
        return nil, err
    }

    req.Header.Add("Authorization", string("Bearer " + token))
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{}
    resp, err := client.Do(req)
    defer resp.Body.Close()
    if err != nil {
        return nil, err
    }

    body, err := ioutil.ReadAll(r
    if err != nil {
        return nil, err
    }

    return body, nil
}
```

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

Akatsuki

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

```go
func getItems(token string) (*ItemList, error) {
    body, err := getRequest("items", token)
    if err != nil {
        return nil, err
    }

    itemResponse := new(ItemList)
    fmt.Println("getItemsResponse", string(body))
    jsonParseError := json.Unmarshal(body, &itemResponse)
    if jsonParseError != nil {
        return nil, jsonParseError
    }

    return itemResponse, nil
}
```

```go
type ItemList struct {
    Data []Item
}

type Item struct {
    Id string
    Attributes ItemDetail
}

type ItemDetail struct {
    Name string
}
```

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

Akatsuki

- 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 })
```

```
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

Akatsuki

- 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

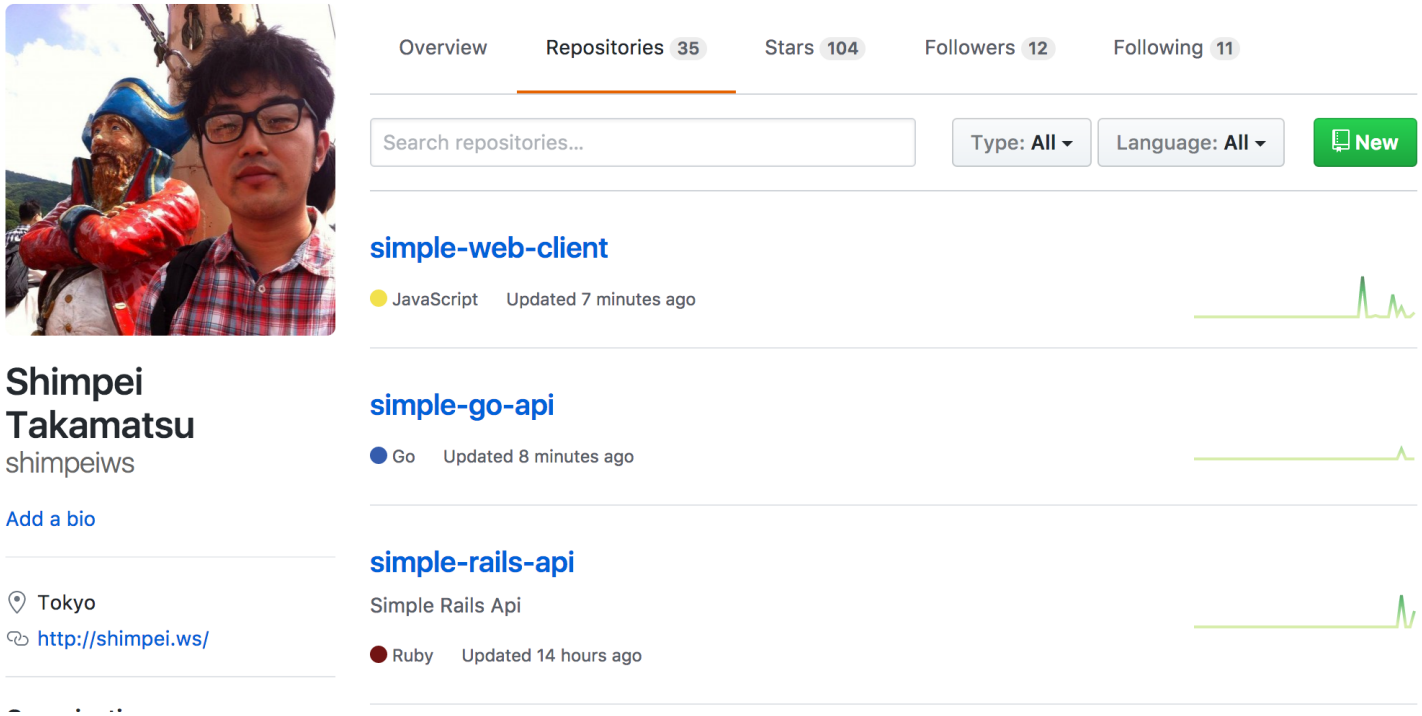
Akatsuki

- **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

Akatsuki

- 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**



まとめ

Akatsuki

- 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!!!

Akatsuki

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

🙏🙏🙏🙏