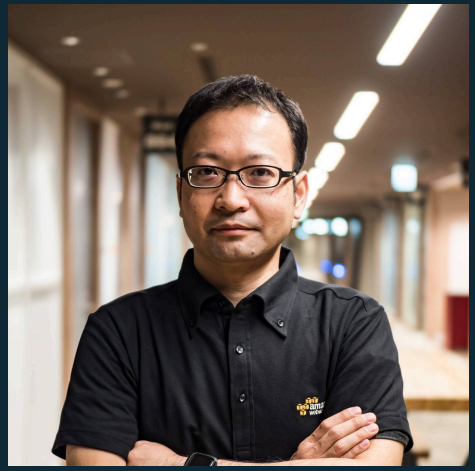
【AWS Black Belt Online Seminar】
AWS Lambda@Edge

アマゾンウェブサービスジャパン株式会社
ソリューションアーキテクト 藤原 吉規

2018.02.21

自己紹介



藤原 吉規 (ふじわら よしのり)

西日本担当 ソリューション アーキテクト

- AWS 大阪オフィスにいます
- 関西のビジネスチャットスタートアップ企業で 6 年間 AWS を活用
- Edge 系サービスを担当
- AWS サムライ 2013
- 好きな AWS サービス: **AWS サポート**

AWS Black Belt Online Seminarとは

AWSJのTechメンバがAWSに関する様々な事を紹介するオンラインセミナーです

【火曜 12:00～13:00】
主にAWSのソリューションや
業界カットでの使いどころなどを紹介
(例：IoT、金融業界向け etc.)

【水曜 18:00～19:00】
主にAWSサービスの紹介や
アップデートの解説
(例：EC2、RDS、Lambda etc.)

※開催曜日と時間帯は変更となる場合がございます。最新の情報は下記をご確認下さい。
オンラインセミナーのスケジュール&申し込みサイト https://aws.amazon.com/jp/about-aws/events/webinars/

内容についての注意点

- 本資料では2018年2月21日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

Agenda

- なぜ Edge で Web サイトを動かすのか？
- Lambda@Edge とは？
- Lambda@Edge がどのように役立つのか？

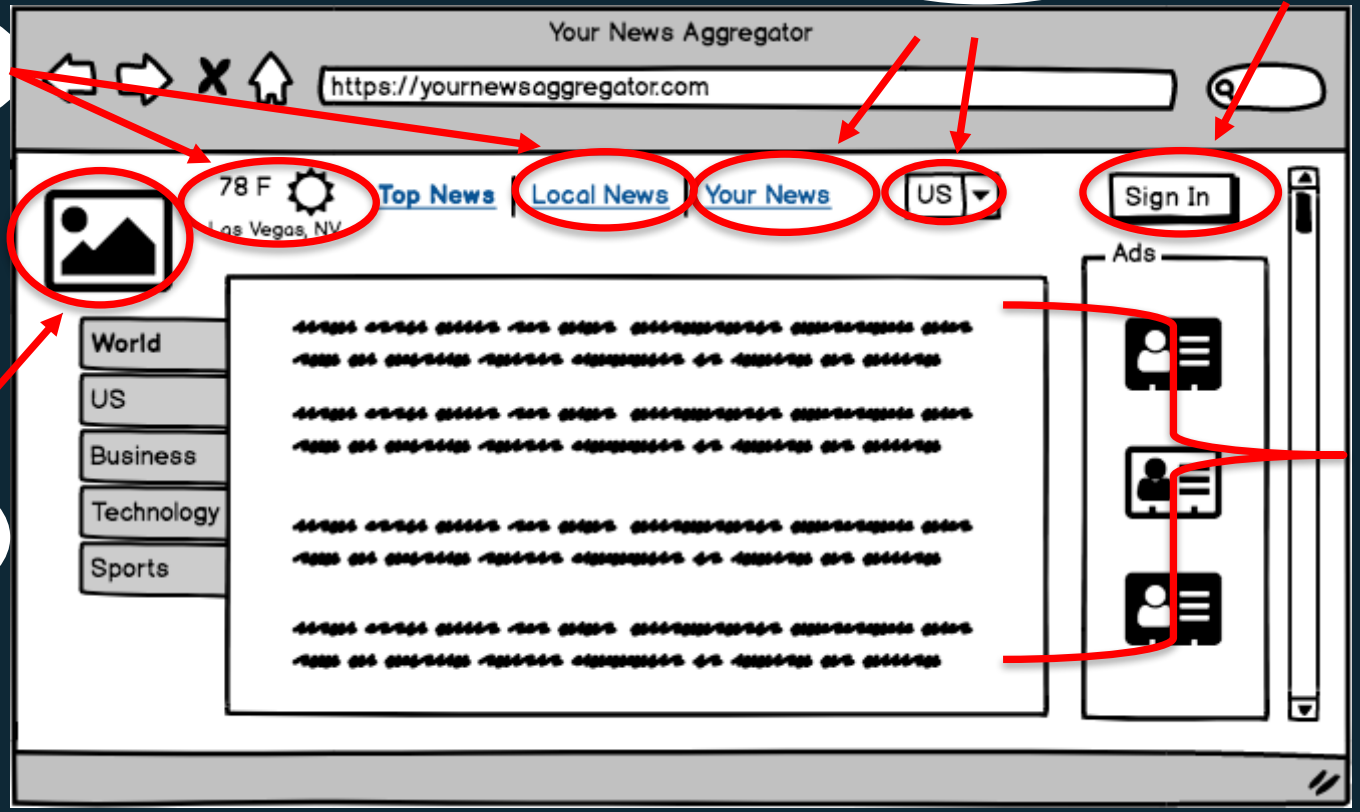
はじめに。。。

- あなたは画期的なアイディアを持っていて

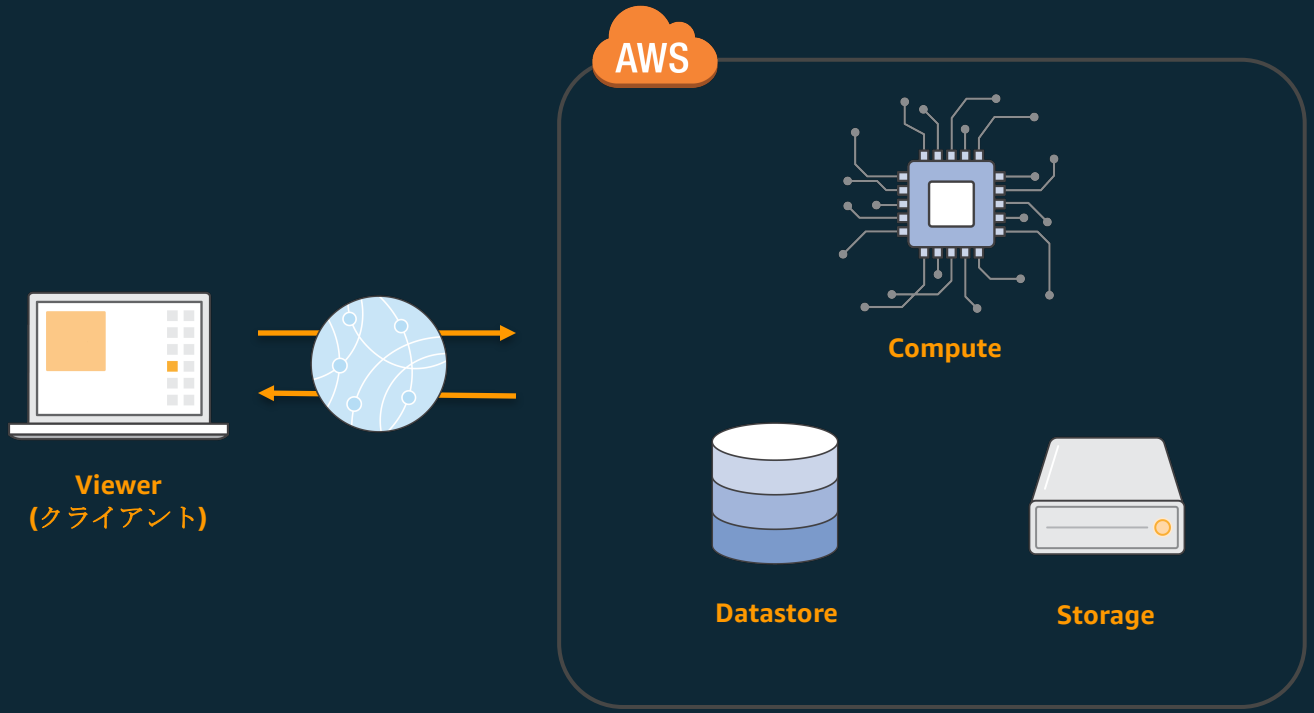
- MVP で Web サイトを構築することに決め

- AWS をプラットフォームに選択しました

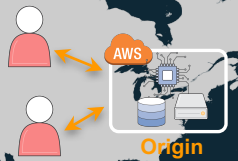
例: ニュース配信サイト



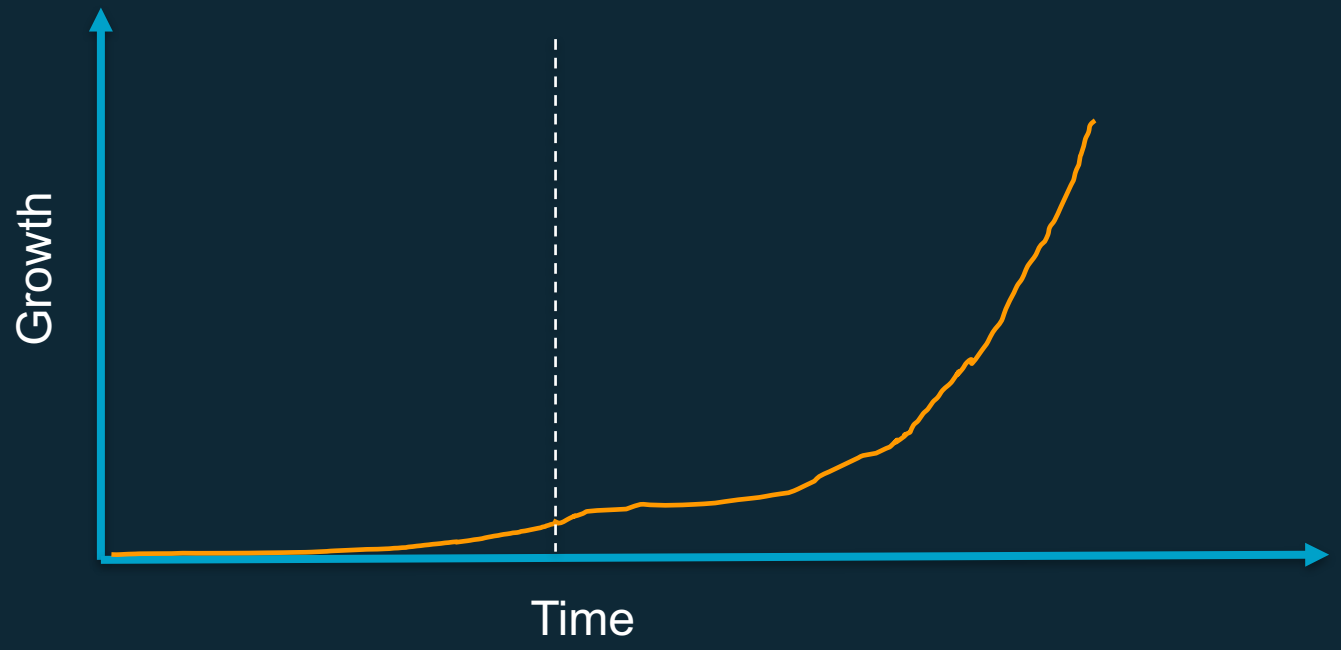
シンプルなアーキテクチャ



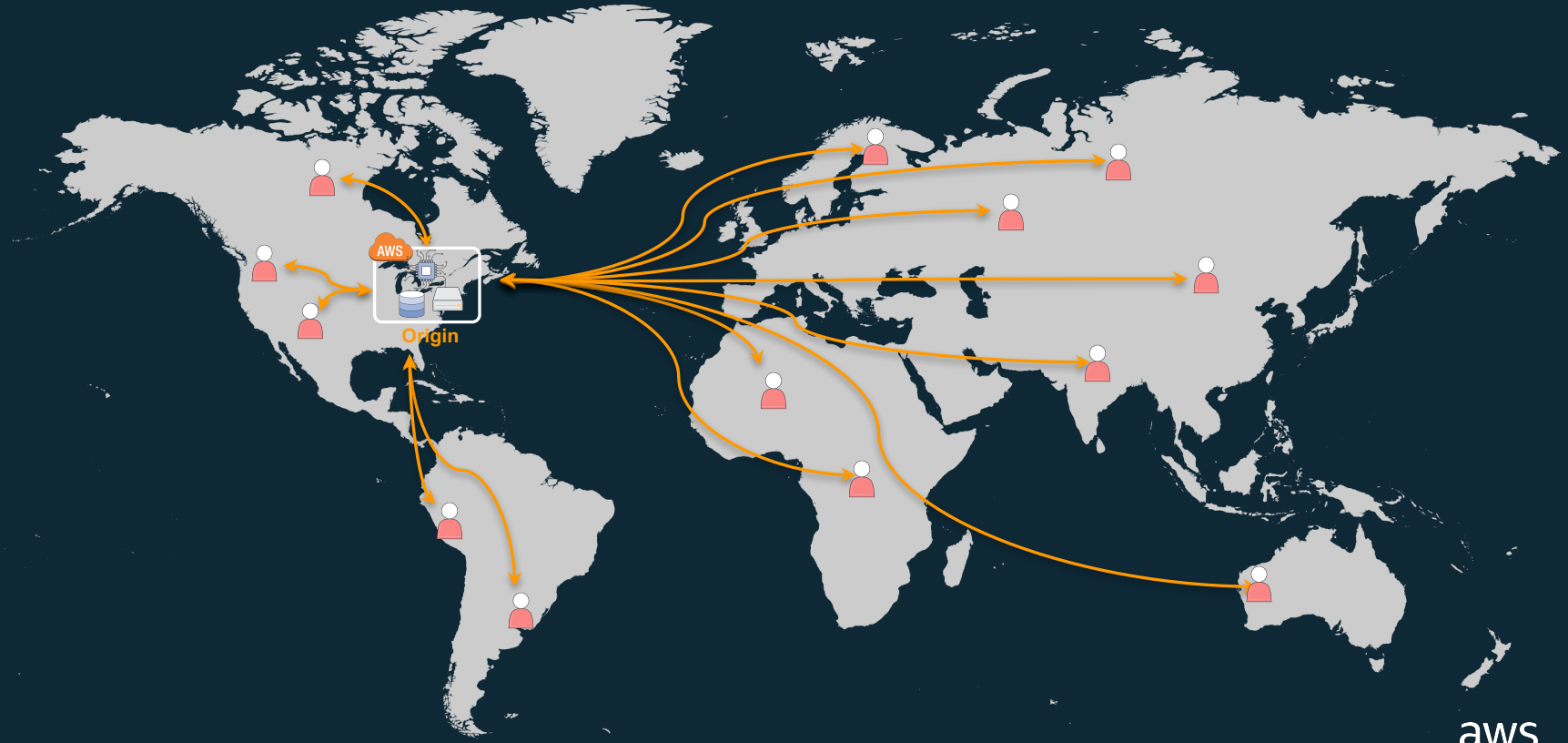
シンプルなアーキテクチャ



その後、サイトがヒットして成長フェーズへ



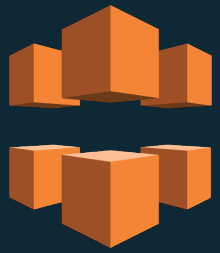
そして、今では。。。



新たなチャレンジ

- 増加し続けるリクエストに対応
  - スケールするインフラストラクチャ
  - 運用上の複雑さを管理
- もちろん、優れたユーザーエクスペリエンスを提供し続ける
  - 速いページ読み込み
  - 豊富な機能セット

Lambda@Edge



Amazon CloudFront

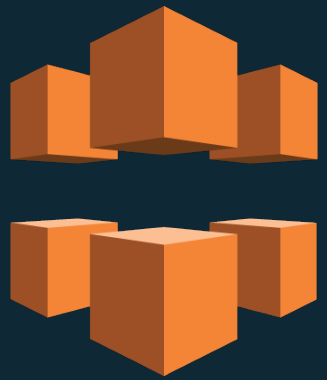
AWS Lambda

Lambda@Edge

aws

Amazon CloudFront

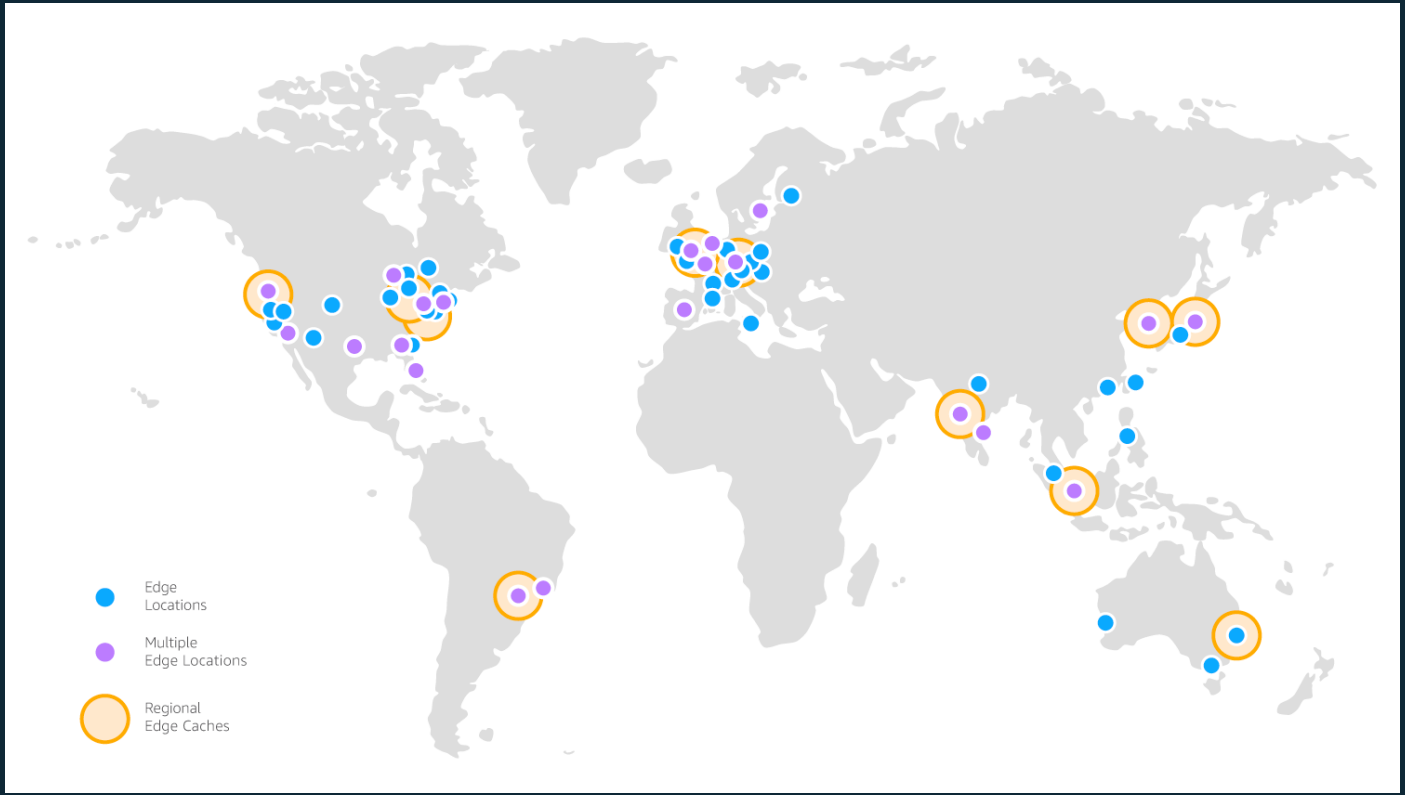
Content Delivery Network



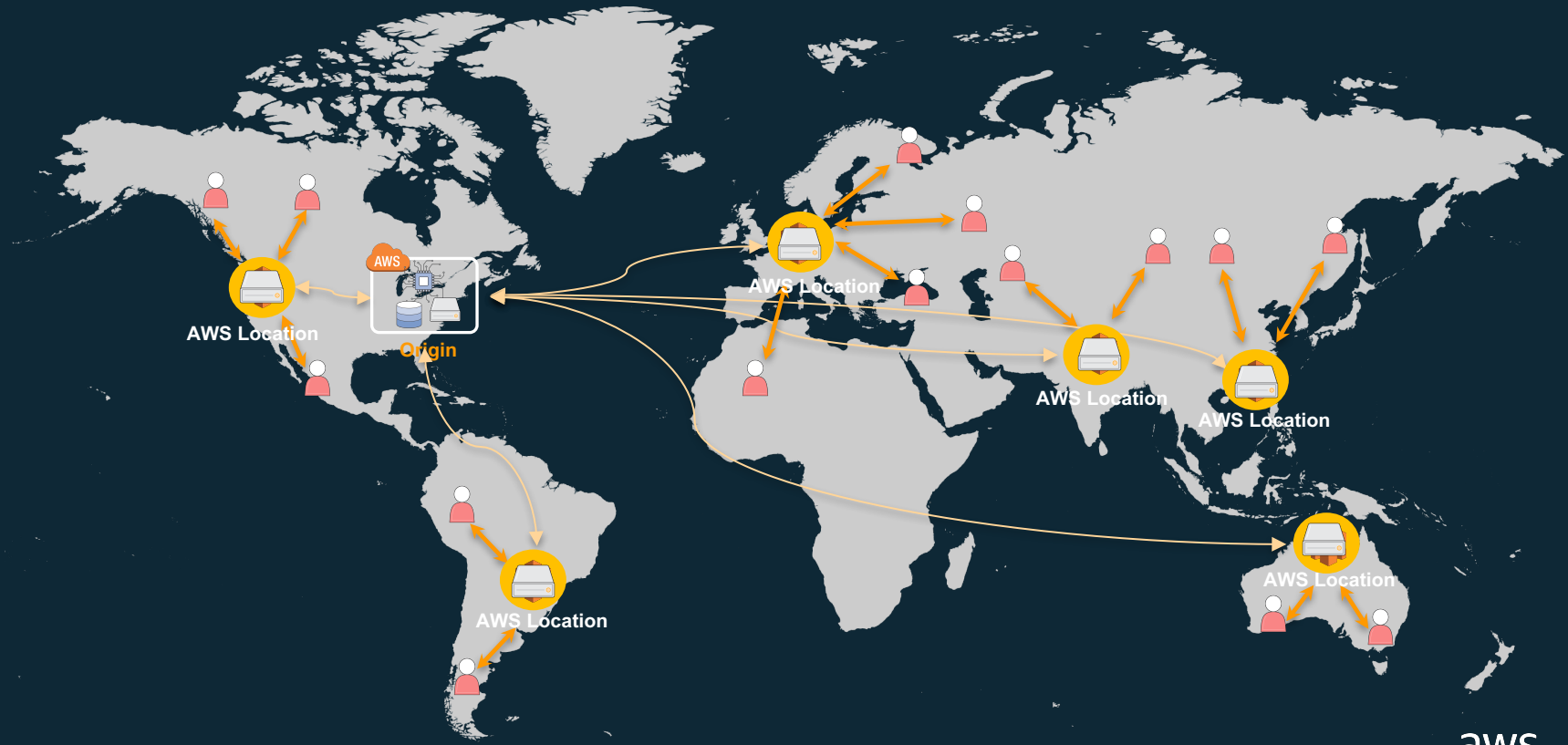
aws

Amazon CloudFront Global Content Delivery Network

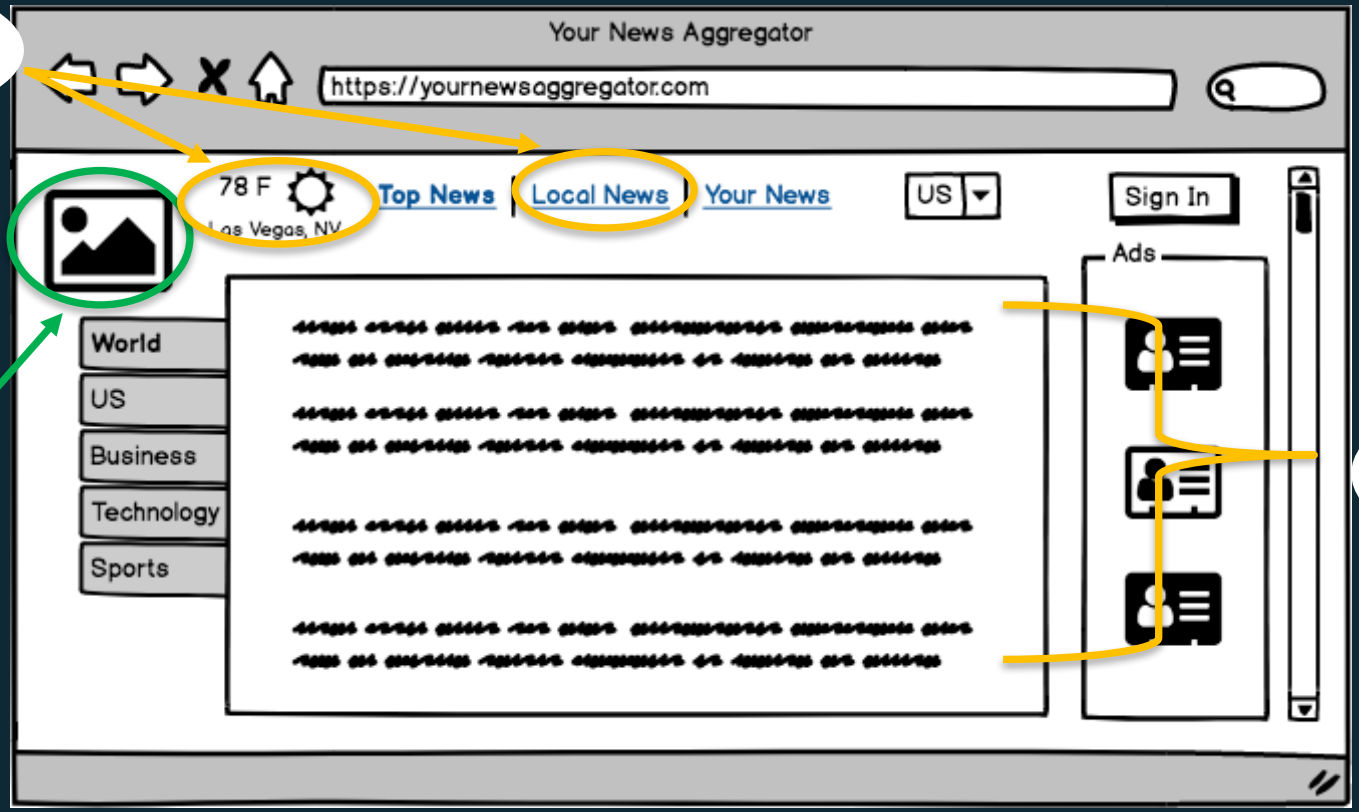
117 PoPs (103 エッジロケーション + 11 リージョナルエッジキャッシュ)



Amazon CloudFront



CloudFront によるコンテンツアクセラレーション



AWS Lambda
Serverless Compute

aws

AWS Lambda

完全に自動化
された管理

自動
スケーリング

利用に応じた
支払い

組み込みの
耐障害性

aws

AWS Lambda@Edge

完全に自動化
された管理

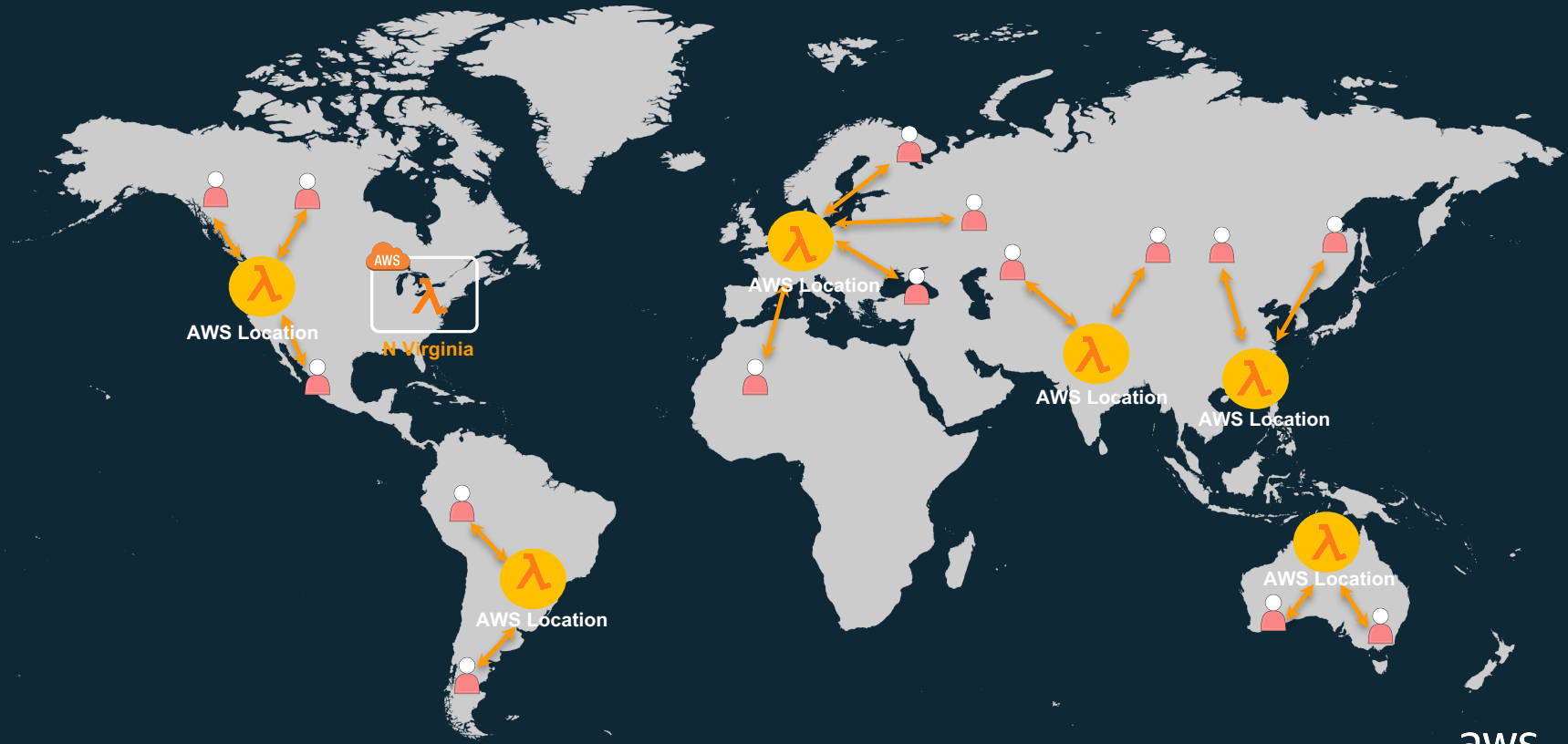
自動
スケーリング

利用に応じた
支払い

組み込みの
耐障害性

グローバル
分散

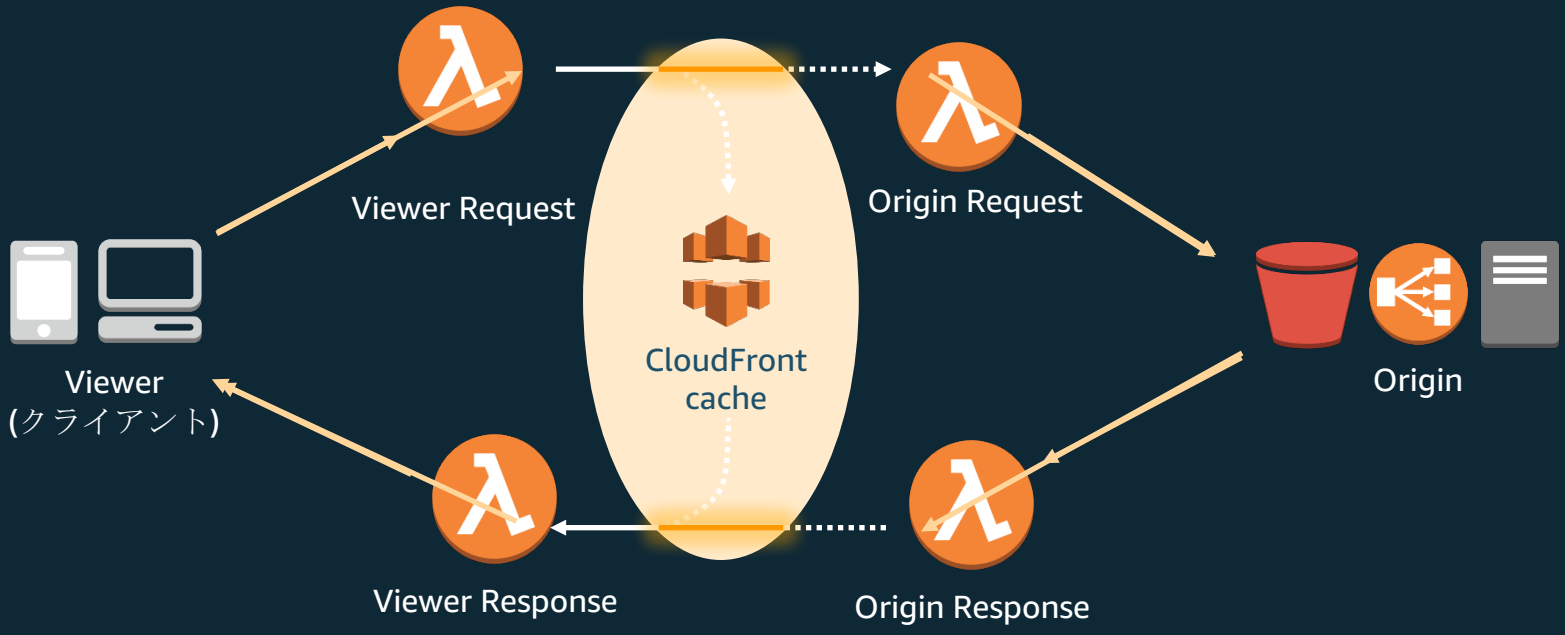
一度 Lambda 関数を書けば、グローバルで実行可能



Lambda@Edge – Deep dive

aws

Lambda@Edge Events



https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/lambda-cloudfront-trigger-events.html

aws

Lambda@Edge のプログラミングモデル

イベント・ドリブン

- 関数はイベントに関連付けられる
  - viewer-request -> my_function:1
- 関数はイベント発生時に実行される
  - viewer-request は CloudFront がリクエストを受信した時に実行される
- 関数は入力イベントの内容を受け取って実行される
  - my_function:1 はリクエストオブジェクトを受け取って実行される
- 関数は呼び出し元に変更した結果を返す必要がある
  - callback(null, request)

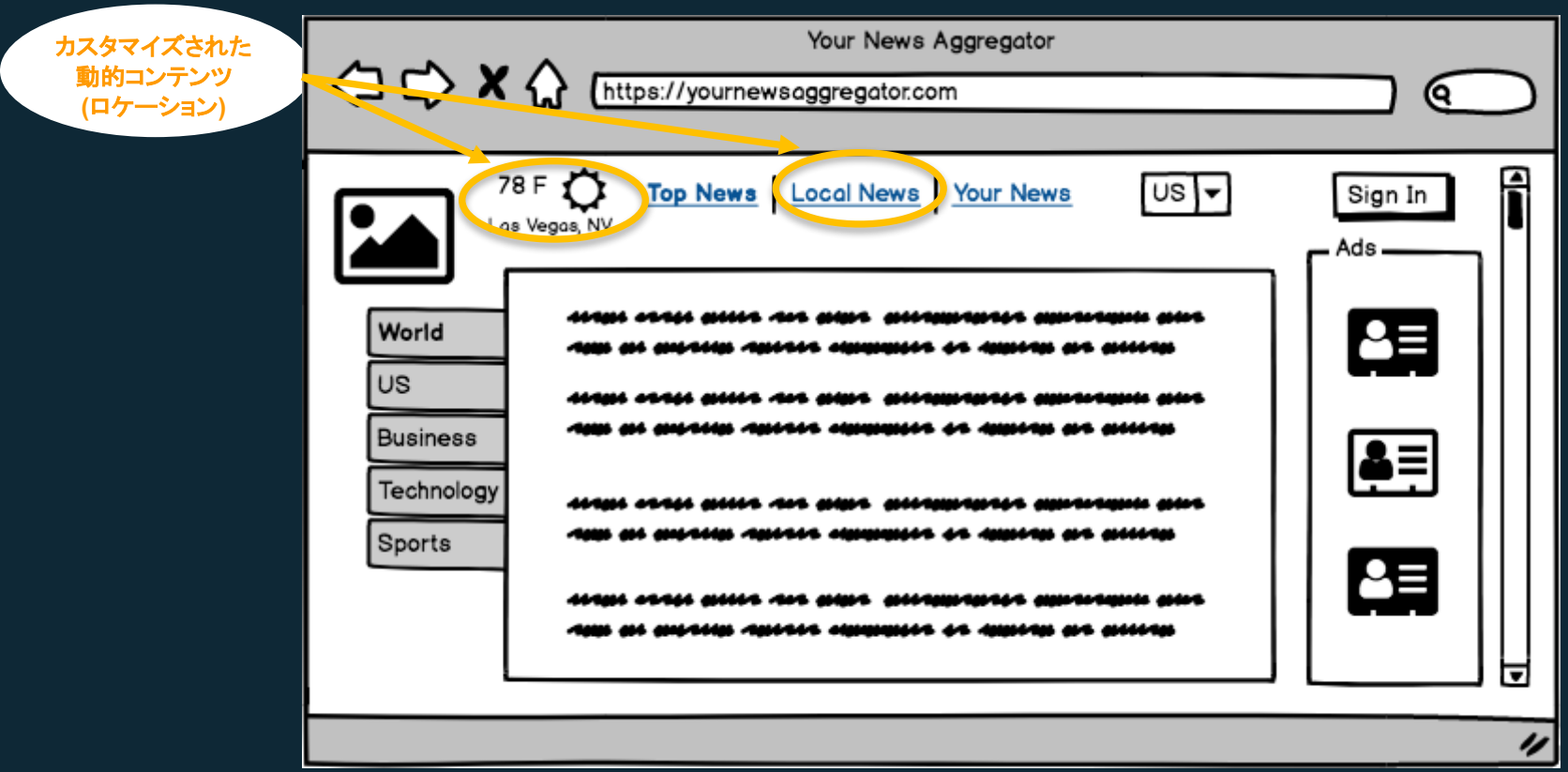
aws

Lambda@Edge のプログラミングモデル

```
exports.handler = (event, context, callback) => {

    /* viewer-request and origin-request events
     * have the request as input */
    const request = event.Records[0].cf.request;

    /* viewer-response and origin-response events
     * have the response as input */
    /* const response = event.Records[0].cf.response; */

    /* Do the processing - say add a header */

    /* When I am done I let CloudFront what to do next */
    callback(null, request);
}
```


aws

Lambda@Edge によるカスタマイズ



カスタマイズ – 入力

パーソナライズ以外のカスタマイズ – リクエスト属性

- 地理的: ローカル/地域/国
  - 天気、最新の交通情報、ニュース、イベント
- デバイスタイプ
  - Mobile / Desktop
  - 解像度: HD / 720p
- User Agent
  - クローラー / 実際のユーザ
- Referer

- パーソナライズ
  - ユーザID + そのユーザに関連する情報

カスタマイズ – 設定

オリジンベース

- CloudFront cache を使用 – URL, Header, クエリパラメータ
- オリジンの動的な選択

Serverless

- CloudFront cache を使用 – URL, Header, クエリパラメータ
- Edge でコンテンツを生成

カスタマイズ – 例

URL の変更

- Viewer(クライアント) アクセス元の国ごとに異なるコンテンツを配信
  - 例: “/index.html” + CloudFront-Viewer-Country -> “/jp/index.html” or “/us/index.html” or “/tw/index.html”)

- 同じディストリビューションから異なる web assets を配信
  - 例: Host header を使用して URL をリライト
  - “/index.html” + “Host: foo.com”-> URL: “/foo_com/index.html”
  - “/index.html” + “Host: bar.com” -> URL: “/bar_com/index.html”

- デバイスタイプごとに異なるコンテンツを配信
  - 例: “/index.html” + CloudFront-Is-Mobile-Viewer -> URL: “/mobile/index.html” or “/desktop/index.html”

カスタマイズ – 例

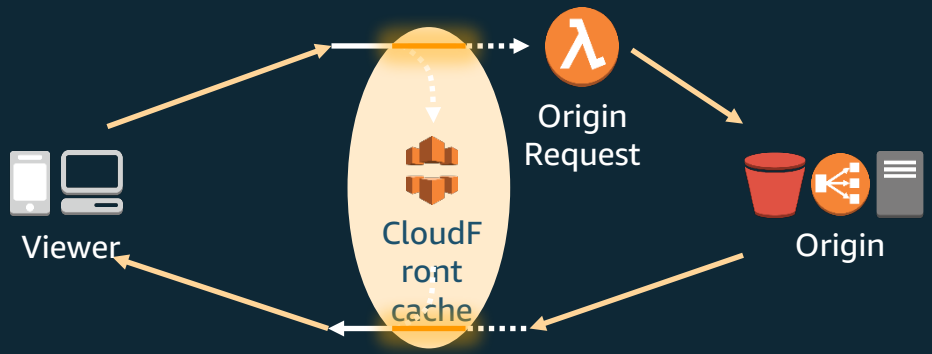
ヘッダの追加/変更/削除

- セキュリティヘッダ
  - 例: HSTS, CORS ヘッダを全てのレスポンスに追加
- S3 オブジェクトに Cache-control ヘッダを設定
  - 例: 一度に大きなグループに対して設定を行う
  - /images/* or /*.jpg
- オリジンに正しいクライアントIPアドレスを提供する
  - 例: オリジンが判断可能なヘッダ (True-Client-IP) に X-Forwarded-For をコピーする

aws

例: デバイスタイプごとのカスタマイズ

クライアントリクエスト https://example.com/index.html



- CloudFront-Is-Mobile-Viewer: true -> https://example.com/mobile/index.html
- その他は全て https://example.com/desktop/index.html へリクエスト
- コンテンツは高い確率でキャッシュ可能なため origin-request event でロジックを実行する
- 正しくコンテンツをキャッシュするために、CloudFront-Is-Mobile-Viewer ヘッダーを転送する設定を行う

例: デバイスタイプごとのカスタマイズ

```
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    /* Direct to a different part of the website based on
     * the device type */
    const desktopPath = '/dektop';
    Const mobilePath = '/mobile';

    if (headers['cloudfront-is-desktop-viewer'] && headers['cloudfront-is-desktop-viewer'][0].value
=== 'true') {
        request.uri = desktopPath + request.uri;
    } else if (headers['cloudfront-is-mobile-viewer'] && headers['cloudfront-is-mobile-
viewer'][0].value === 'true') {
        request.uri = mobilePath + request.uri;
    }

    console.log(`Request uri set to "${request.uri}"`);
    callback(null, request);
}
```

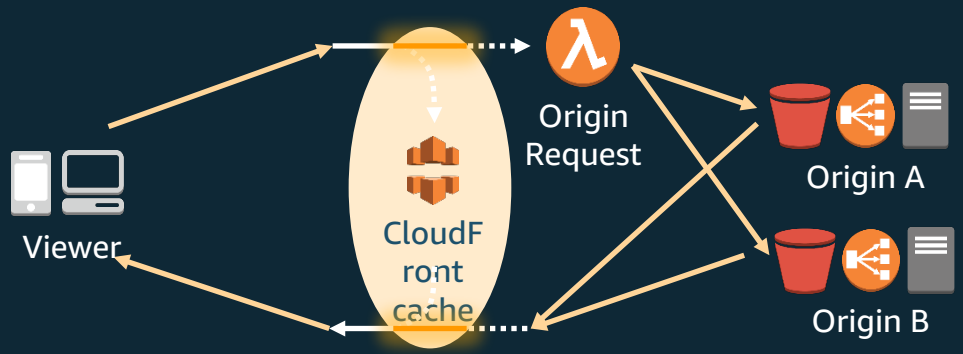

aws

オリジンの動的な選択によるカスタマイズ

- 複数オリジンの設定
  - レイテンシベース: Viewer(クライアント) に最も近いオリジンを選択
  - 複数オリジン間でロードバランシング
- オリジン側で変更の公開を制御
  - 新機能の A/B テスト
  - オリジンの Blue/Green デプロイ
- オリジン間の移行
  - オリジンの一部にオンプレミスを含み、徐々にクラウドに移行する
- SEO: 検索エンジン最適化
  - 実際のユーザーとクローラー用のトラフィックを別々のオリジンから提供する

オリジンの選択 – A/B テスト

例: 新機能をテスト、新機能はオリジンの一方にのみデプロイされている



関数にて:

1. リクエストがアクティブセッションであるかを確認する (Cookie を使用)
2. アクティブセッションの場合は、 Cookie の値に基いてオリジンを設定する
3. 新しいセッションの場合は、A または B のどちらを表示するか決定し、オリジンを設定する

オリジンの動的な選択

- オリジンはリクエストイベントの一部として存在
  - `event.Records[0].cf.request.origin`
- 変更したオリジンはcallbackに返すリクエストオブジェクトに含める

```
"s3": {
 "domainName": "green-bucket.s3.amazonaws.com”,
 "path": "/originPath",
 "authMethod": "origin-access-identity",
 "region": "us-east-1",
 "customHeaders": {
    "my-custom-origin-header": [
       {
        "key": "My-Custom-Origin-Header",
        "value": "test-value”
       }
    ]
  }
}
```


aws

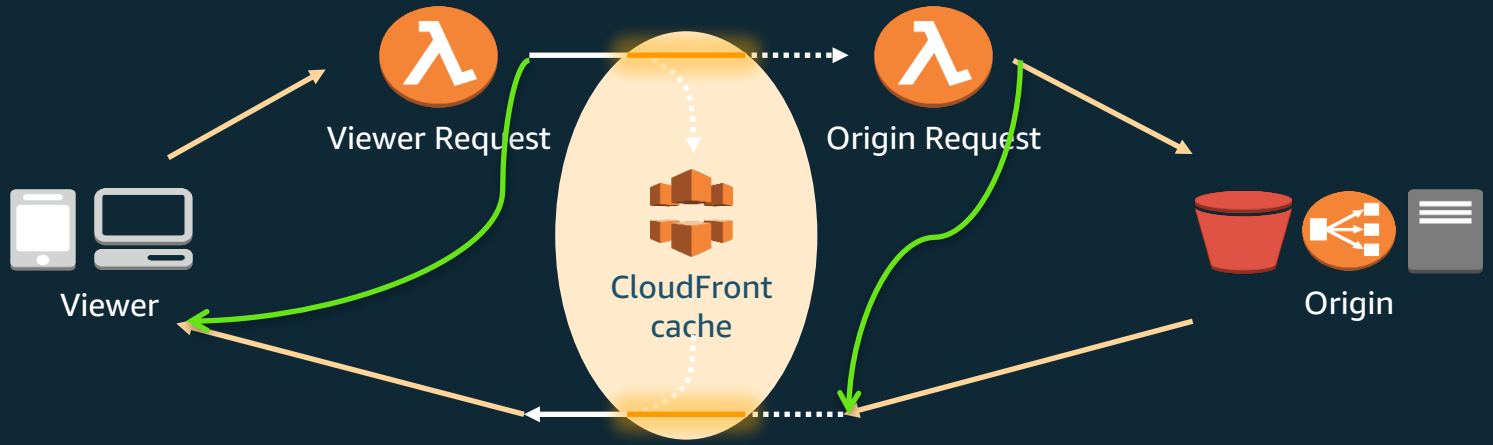
例 – A/B テスト

```
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    desiredOrigin = decide(request);

    /* Set custom origin fields*/
    request.origin = {
                custom: {
                    domainName: desiredOrigin,
                    port: 443,
                    protocol: 'https',
                }
    };
    request.headers['host'] = [{ key: 'host',
                                 value: desiredOrigin }];

    callback(null, request);
};
```

```
function decide(request) {
    if (request.headers[‘my-session-cookie’]) {
        cookie = request.headers[‘my-session-
cookie’].value;
        return decodeOrigin(cookie);
    } else {
        return chooseOrigin(request);
    }
};
```


aws

レスポンス生成



- Edge ロケーションでレスポンスを生成
- viewer-request または origin-request からレスポンスを返却
- リクエストは Lambda@Edge で全て処理されるため、オリジンに到達しない

レスポンス生成

- Edge ロケーションでレスポンスを生成
- viewer-request または origin-request イベントで実行可能
- 関数内で全ての HTTP レスポンスを生成する
- このレスポンスは callback に含める

```
callback(null, response);
```

- リクエストはオリジンに到達しない

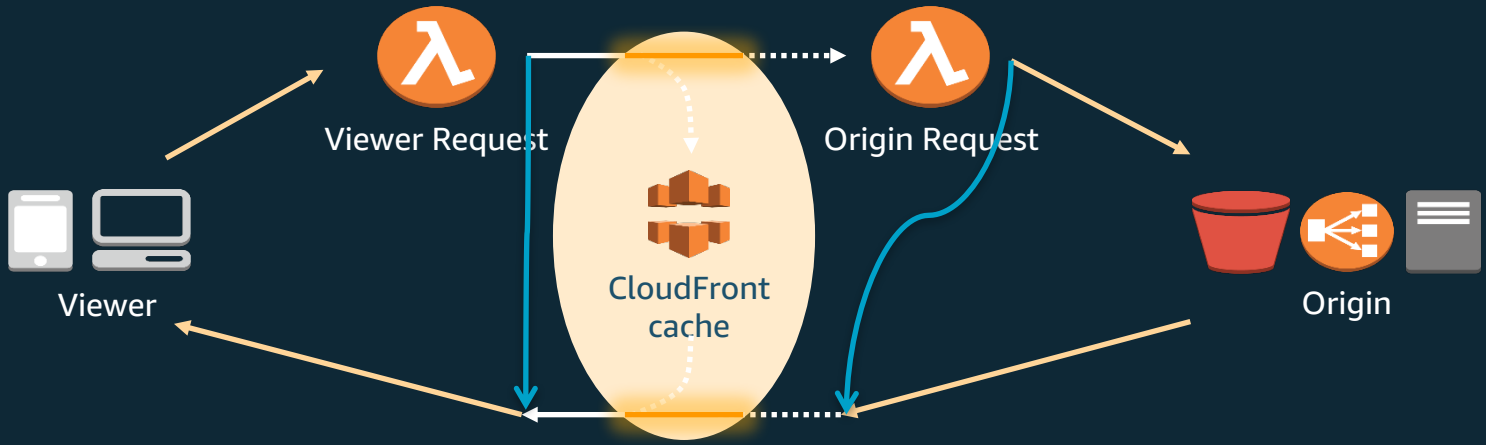
aws

レスポンス生成を使用したカスタマイズ

- ヘッダのみ生成
  - Redirect
  - 認証/認可に外部の source を利用
- ヘッダ / Body 生成
  - サイト構成イメージなど高い確率でキャッシュ可能なコンテンツ
  - Viewer ごとにユニークなレスポンス
  - リクエスト入力を利用
    - 例: 天気アプリ – ランディングページ
      - 表示地域のリストはユーザごとに異なり、リストがリクエストに含まれる
      - 地域ごとの天気情報は高い確率でキャッシュ可能
  - リクエストに含まれない、ユーザー固有のデータを使用
    - 例: 私の Prime Music ページ
      - 私のアルバムリスト
      - おすすめのプレイリスト
      - データストアから取得

aws

例 – Redirect



- viewer-request または origin-request イベント
- 未認証ユーザーをログインページにリダイレクト
    - viewer-request 関数で Cookie を検証
    - Cookie が期限切れもしくは存在しない場合は、ログインページへリダイレクト

例 – Redirect

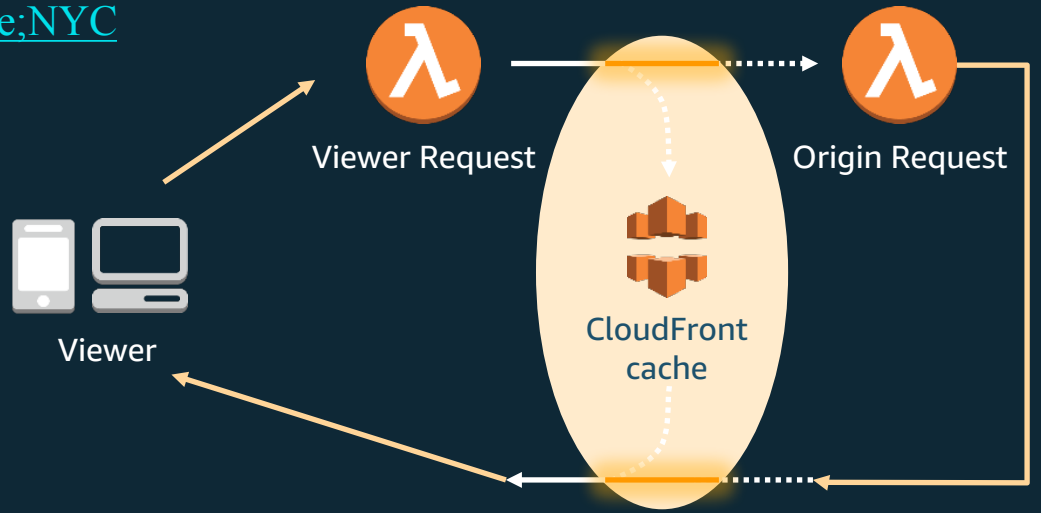
```
'use strict';

exports.handler = (event, context, callback) => {
    /*
     * Generate HTTP redirect response with 302 status code and Location header.
     */
    const response = {
        status: '302',
        statusDescription: 'Found',
        headers: {
            location: [{
                key: 'Location',
                value: 'http://mydomain.com/login.html',
            }],
        },
    };
    callback(null, response);
};
```


aws

例 – コンテンツ統合

例: 天気アプリのランディングページ
クライアント: 各ユーザは表示地域のリストを持つ

http://example.com/weather?cities=Seattle;NYC



関数にて:

- URL のパース
- 関連データの取得
- 統合されたレスポンスをクライアントアプリへ送信

例 – コンテンツ統合

```
function getCityForecast(request) {
    return new Promise((resolve, reject) => {
        https.get(request.uri, (response) => {
            let content = '';
            response.setEncoding('utf8');
            response.on('data', (chunk) => { content += chunk; });
            response.on('end', () => resolve({ city: request.city, forecast: content }));
        }).on('error', e => reject(e));
    });
}

const uriSplit = uri.split('/');
const cities = uriSplit[2].split(':');

const forecasts = [];
cities.forEach((cityName) => {
    const cityForecastUri = citiesBaseUri + cityName;
    forecasts.push({ city: cityName, uri: cityForecastUri })
});
```

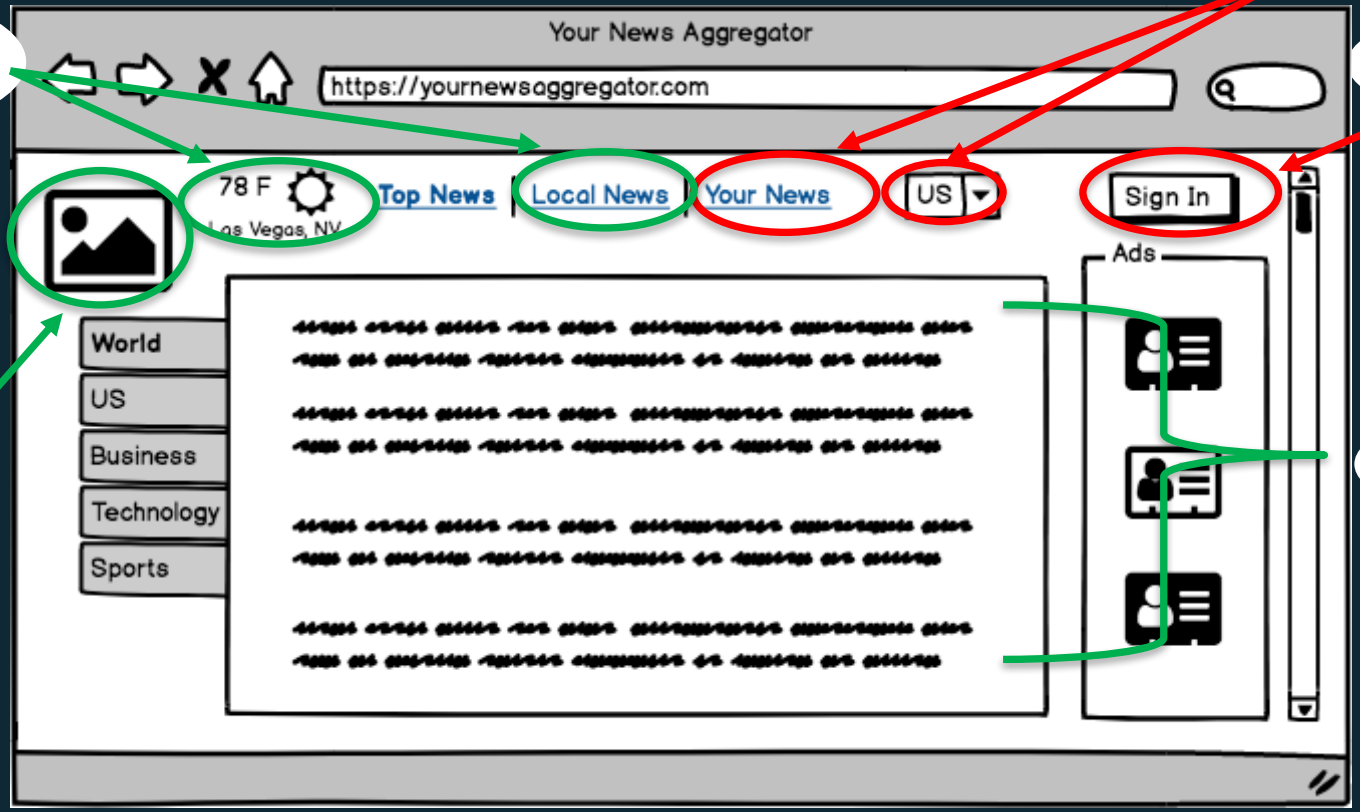

aws

例 – コンテンツ統合

```
Promise.all(forecasts.map(getCityForecast)).then((ret) => {
      console.log('Aggregating the responses:\n', ret);
      const response = {
        status: '200', /* Status signals this is a generated response */
        statusDescription: 'OK',
        headers: {
          'content-type': [{
            key: 'Content-Type',
            value: 'application/json',
          }],
          'content-encoding': [{
            key: 'Content-Encoding',
            value: 'UTF-8',
          }],
        },
        body: JSON.stringify(ret, null, '\t'),
      };

      console.log('Generated response: ', JSON.stringify(response));
      callback(null, response);
```


aws

他にも Edge に移動できるものは？



カスタマイズされた
動的コンテンツ
(ロケーション)

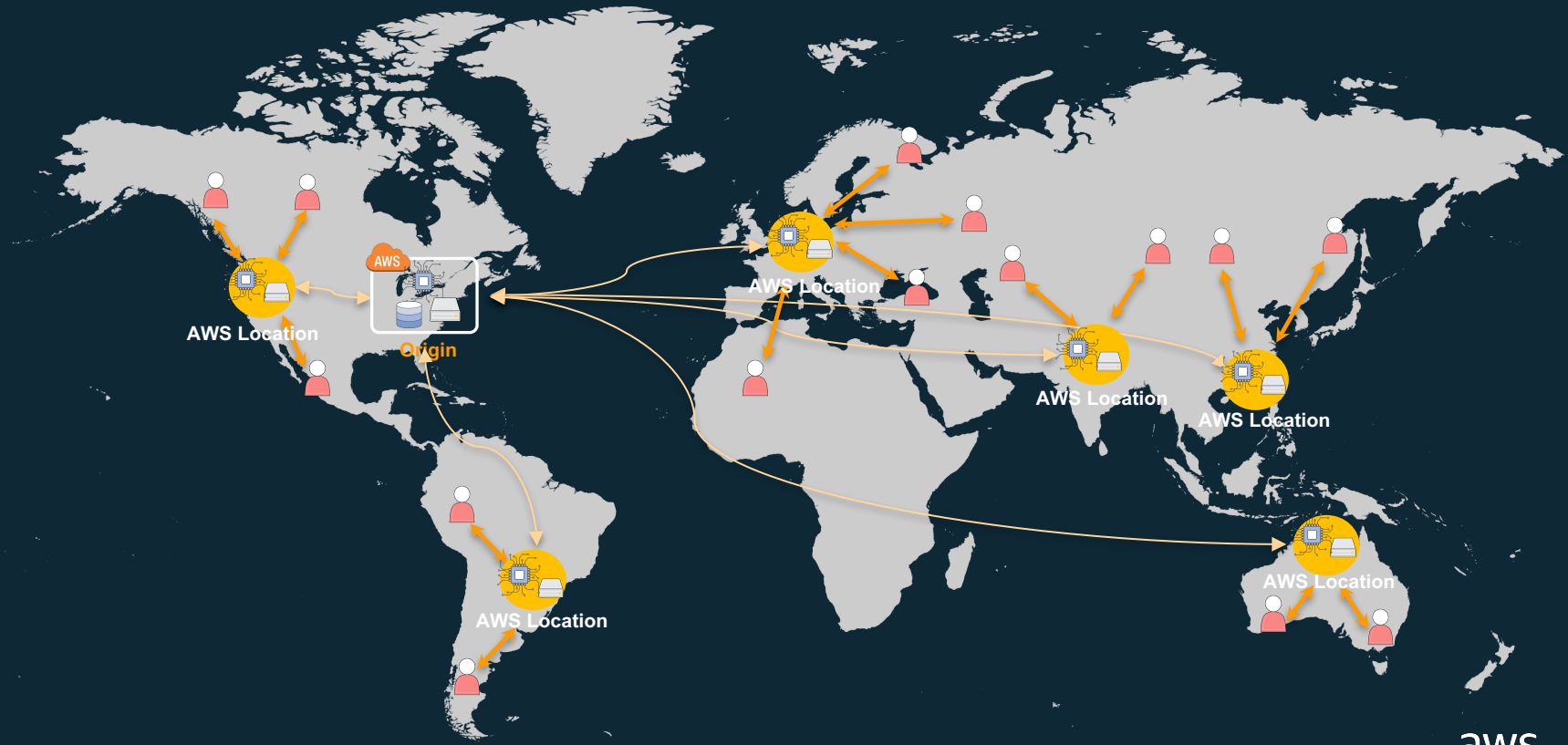
動的なユーザーコンテンツ
(パーソナライズ)

認証

静的コンテンツ
(images,JS,CSS, …)

動的コンテンツ
(News feed)

CloudFront + Lambda@Edge



パーソナライズ

- アイデンティティ: 認証 – カスタマが誰であるかを認識
- 情報: カスタマに関する知識
    - 設定, 閲覧/購買履歴 …
- アイデンティティと情報を組み合わせて、そのユーザにパーソナライズしたコンテンツを生成する

認証

このリクエストを処理するかどうかを決定する仕組み

- 署名付き URL/Cookie を使用した認証 (CloudFront の機能)
- 認証 – Edge の関数で検証
- リモートサーバを使用した認証

関数のデータアクセス方法

- データをコードに含める
  - 使用およびアクセスが容易で高速
  - 読み取り専用、デプロイパッケージの最大サイズに制限される、変更にはデプロイが必要
- CloudFront キャッシュを使用
  - 関数に近い (同じロケーションにある可能性が高い) 、TTL によって定期的にリフレッシュされる
  - 読み取り専用
- AWS DB の利用 (DynamoDB 等)
  - 読み書き可能
  - 関数から離れた一つのリージョンにデータが存在
- Amazon DynamoDB Global Tables
  - マルチリージョン、マルチマスタテーブル
  - 読み書き可能、関数に近いリージョンを利用

aws

コンテンツ生成 – パーソナライズ

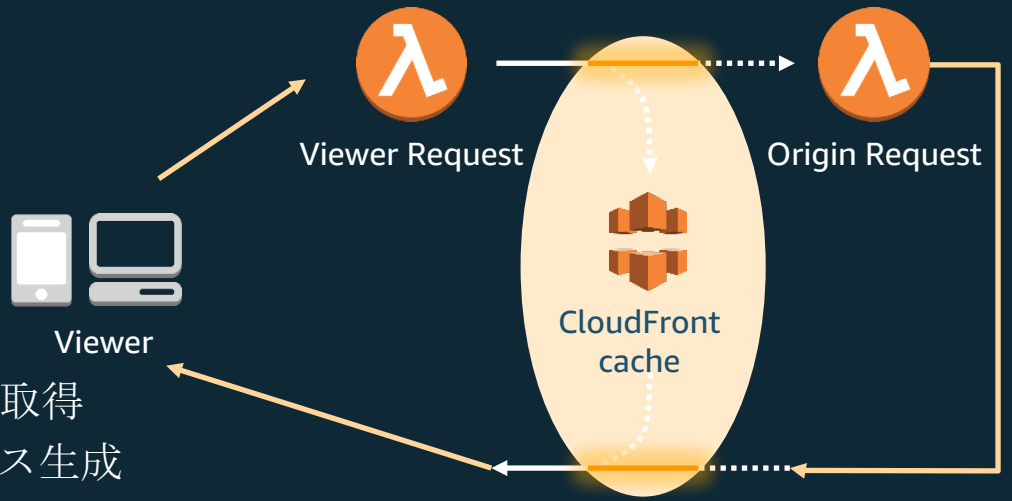
例: ユーザのログイン時に新しい「アイテム」をレコメンデーション

Client:

- https://example.com/index.html
- ユーザはログイン済

関数:

- DynamoDB からユーザの履歴と設定を取得
- 取得されたデータを使用してレスポンス生成
- 例:
    - 閲覧履歴に基づくニュース記事
    - 視聴履歴に基づく音楽および映画のレコメンデーション
    - フォローしている人に基づくレコメンデーション



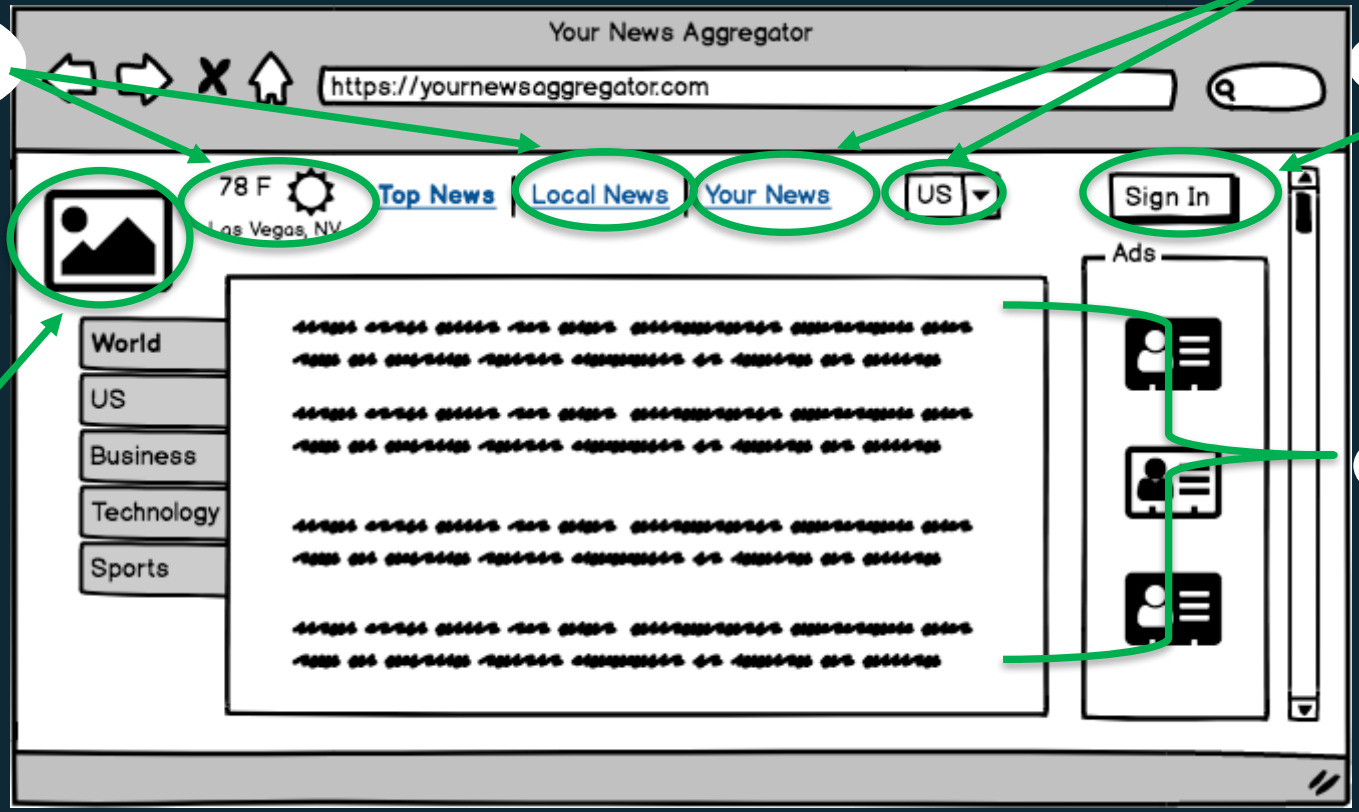
コンテンツ生成 – パーソナライズ

```
/* Access DDB */
var aws = require('aws-sdk');
var ddb = new aws.DynamoDB().DocumentClient({apiVersion: '2012-10-08', region: 'us-east-1'});

/* Get the item based on the user */
function getItems(request, context, callback) {
  const params = {
    TableName: "recommendations",
    Key: { "User": { "S": user } }
  };
  ddb.getItem(params, function(err, data) {
    if (err) {
      console.log(err, err.stack);
      generateErrorResponse(request, callback);
    } else {
      console.log("Received output from ddb: " + data)
      generateResponse(request, data, callback);
    }
  });
}
```


aws

まとめ

Serverless at the Edge



カスタマイズされた
動的コンテンツ
(ロケーション)

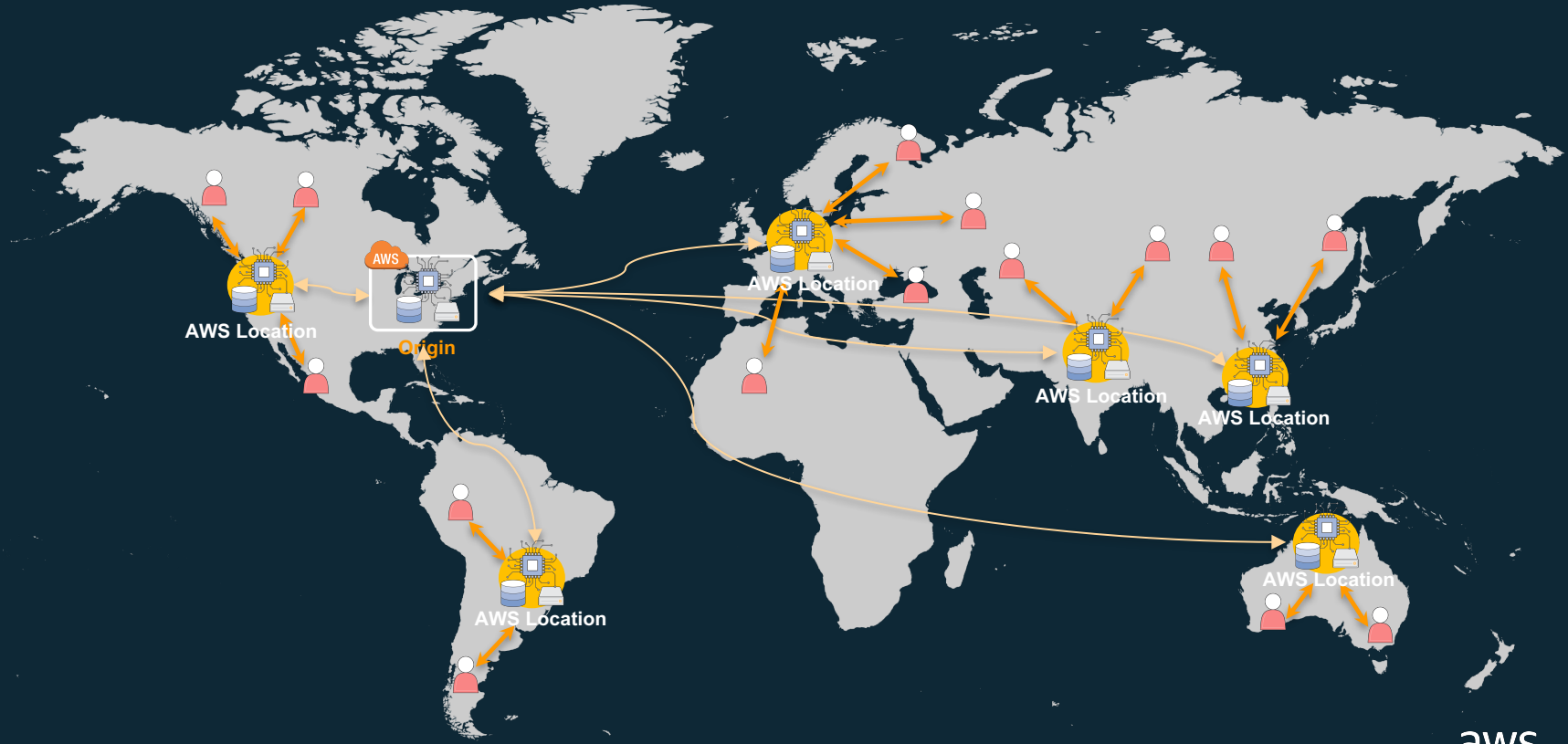
動的なユーザーコンテンツ
(パーソナライズ)

認証

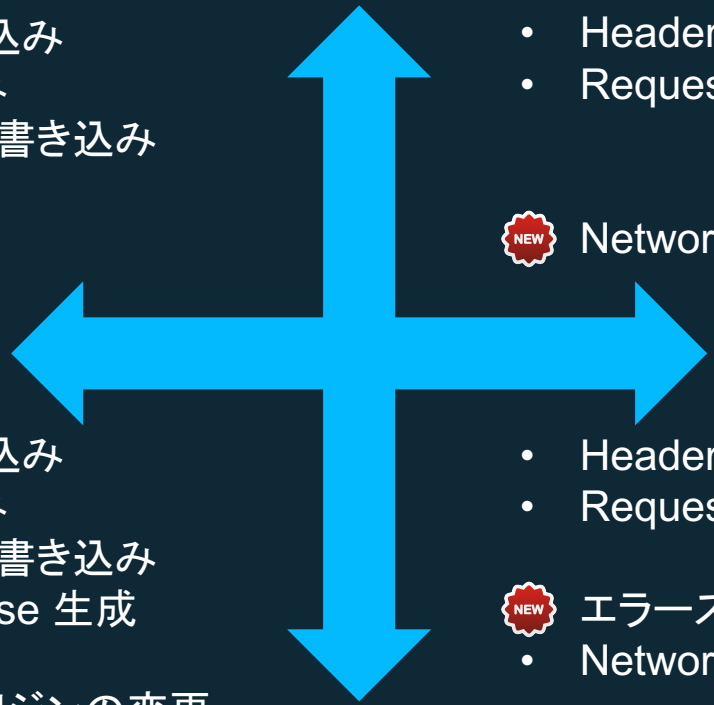
静的コンテンツ
(images,JS,CSS, …)

動的コンテンツ
(News feed)

Serverless at the Edge



CloudFront イベントごとの機能



VIEWER

- Header 読み取り/書き込み
- URL 読み取り/書き込み
- NEW クエリ文字列 読み取り/書き込み
- Response 生成
- NEW Network 呼び出し

- Header 読み取り/書き込み
- Request object 読み取り

- NEW Network 呼び出し

REQUEST

- Header 読み取り/書き込み
- URL 読み取り/書き込み
- NEW クエリ文字列 読み取り/書き込み
- NEW バイナリを含むResponse 生成
- Network 呼び出し
- NEW S3オリジン,カスタムオリジンの変更
- NEW 関数タイムアウト 30 秒

RESPONSE

- Header 読み取り/書き込み
- Request object 読み取り

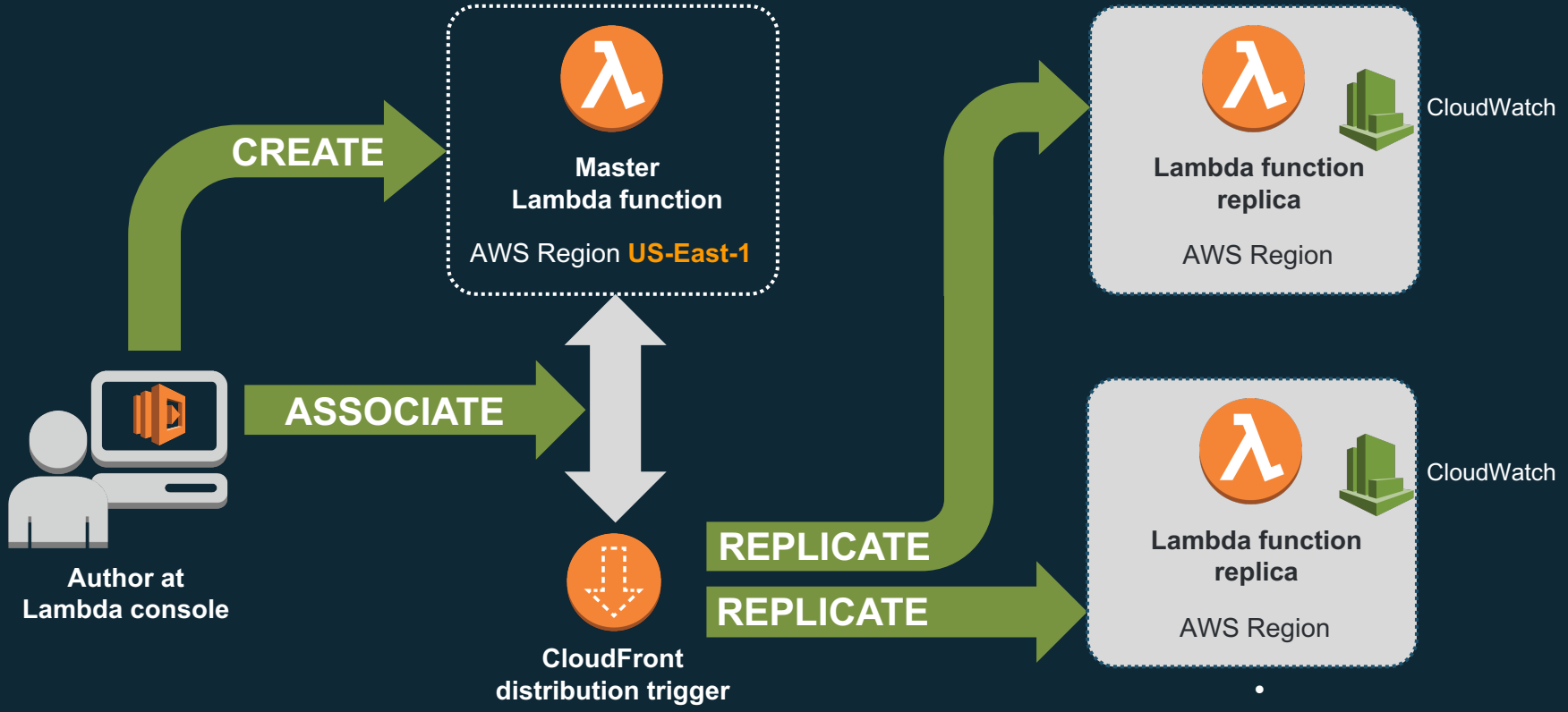
- NEW エラーステータス時の Response 更新
- Network 呼び出し

- NEW 関数タイムアウト 30 秒

ORIGIN

aws

Lambda@Edge 用の Lambda 関数を作成



http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html#lambda-edge-how-it-works

Lambda@Edge 実行環境

	Origin facing	Viewer facing
ランタイム	Node.js 6.10	←
メモリ	Lambdaと同じ (1,536MB)	128 MB
関数タイムアウト	30 秒	5 秒
Lambda 関数および組み込みライブラリの最大圧縮サイズ	50 MB	1 MB
レスポンスサイズ (request events)	1 MB	40 KB
同時実行数のデフォルト (Region毎) ※上限緩和可能	1,000	←
/tmp, 環境変数, DLQ, VPC	使用不可	←

http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-requirements-limits.html

aws

参考資料

全般

- CloudFront と Lambda@Edge の使用 Page
- 関数の例 Page
- よくある質問 Page
- 料金詳細 Page
- AWS BlackBelt Online Seminar 2017 Amazon CloudFront Page
- Building Serverless Websites with Lambda@Edge - CTD309 - re:Invent 2017 Slides / Video

関数のテストとデバッグ

- Lambda 関数の CloudWatch メトリクスと CloudWatch Logs Page
- AWS X-Ray を使用した Lambda ベースのアプリケーションのトレース Page
- SAM ローカルを使用してサーバーレスアプリケーションをローカルでテストする(パブリックベータ) Page

ご参加ありがとうございました

aws