AWS Lambdaで変わるバッチの世界

~ CPUトータル100時間を10分で終わらせるには ~

株式会社ワークスアプリケーションズ
東　卓弥

powered by amazon web services

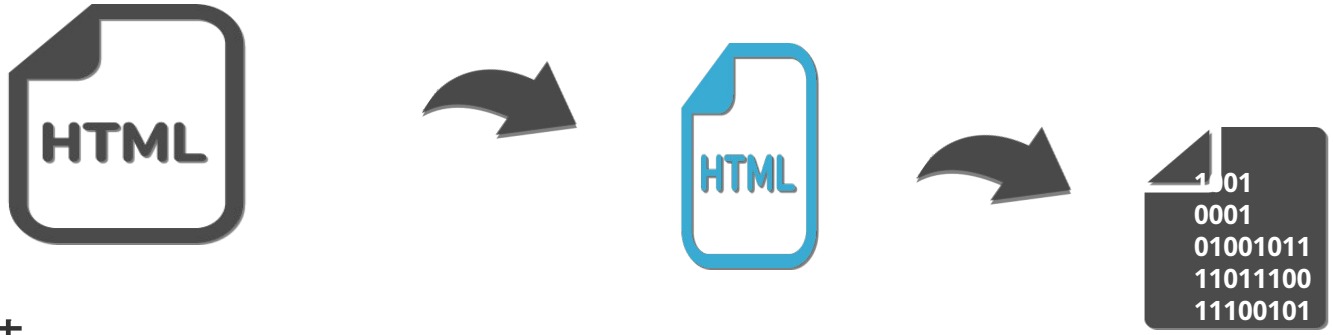
プロフィール

氏名 : 東 卓弥
会社 : 株式会社ワークスアプリケーションズ
所属 : Site Reliability Engineering Div.
役職 : Manager

powered by amazon web services

対象の処理

- 画面の高速描画のための前処理
  - HTML, JS, CSSの最適化
  - HTMLテンプレートの事前コンパイル



- 規模
  - 画面総数：9000弱
  - CPU時間100時間

powered by amazon web services

AWS Lambda 採用の経緯

問題点

長い！　コスト高い！

- CPU時間10時間、1回＄15
- Sparkで2時間。コストとトレードオフ
- 終わらないインスタンス数最適化作業

課題

処理時間　10分

AWS Lambdaってのがあるよ！

powered by amazon web services

AWS Lambdaの特徴

1. スケーリング管理コスト0
2. インスタンス管理コスト0

   選べるRuntime（etc. Java 8, Node.js 4.3, Python 3.6, C#）

3. 100ms単位の課金！

   処理した時間のみ

   流動的な分散数と処理時間にマッチ！

powered by amazon web services

AWS Lambda利用のための課題

処理フローの整理

処理時間上限 5min

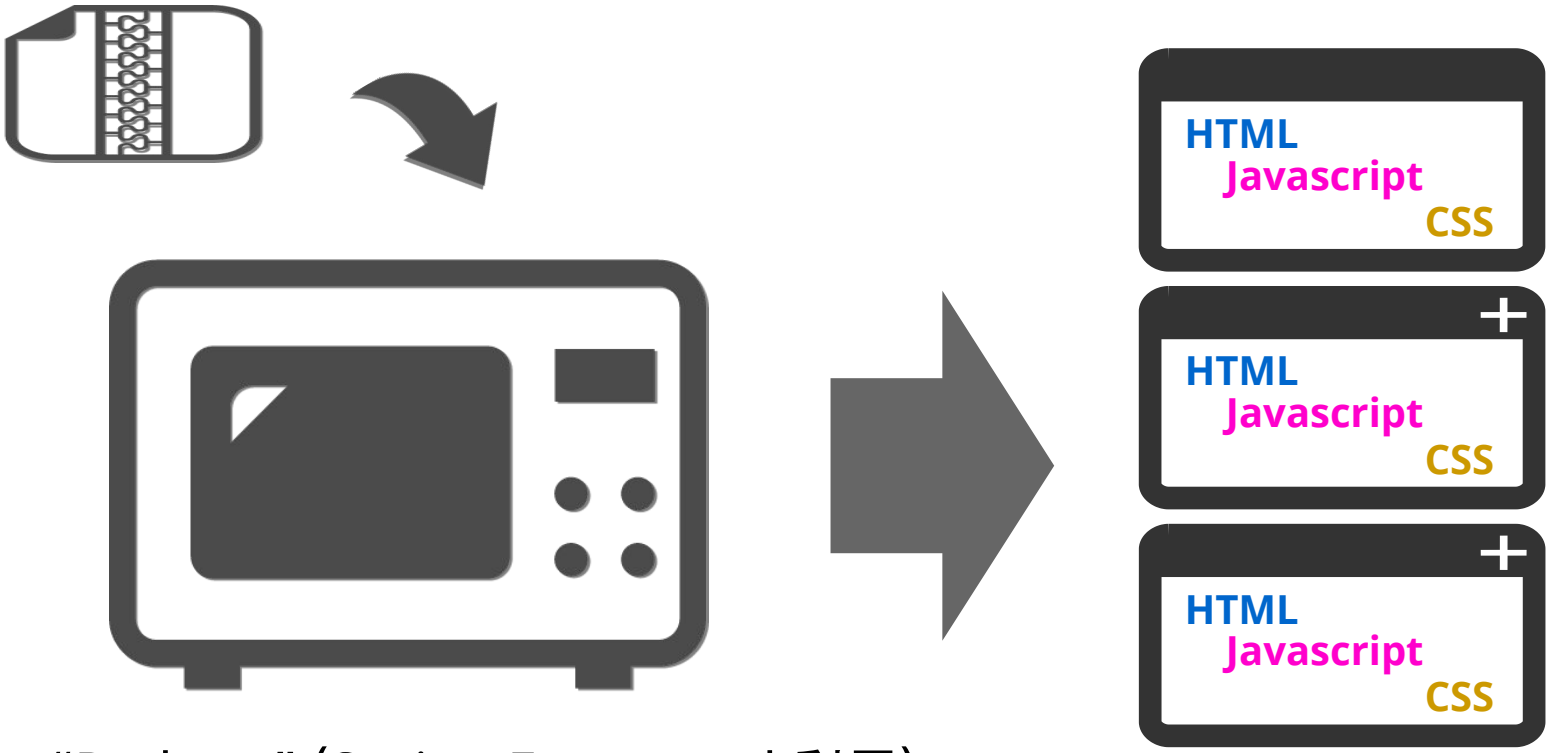
メモリ上限 1.5G

出荷

運用（初期設定）

powered by amazon web services

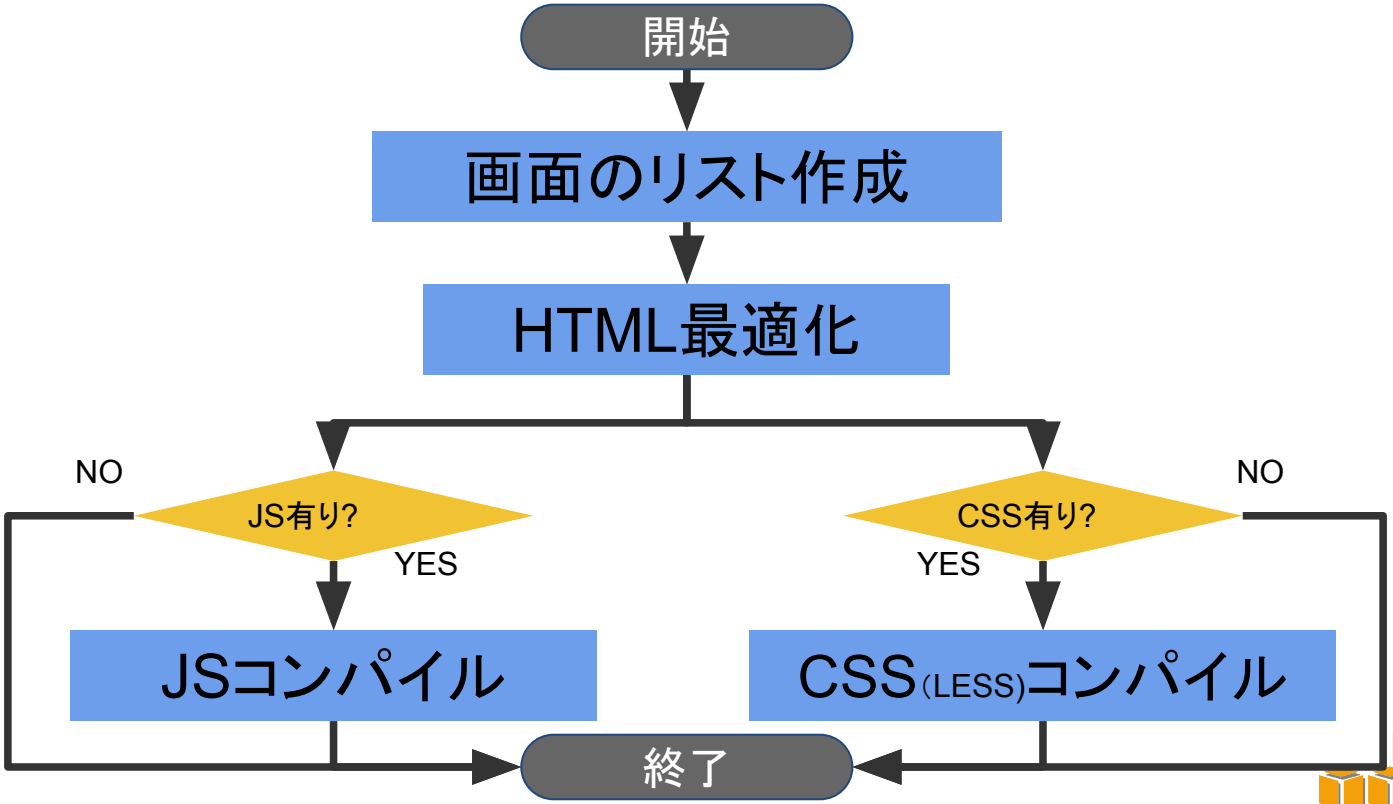
以前の処理



“Preheat” (Spring Framework利用)

powered by amazon web services

フローの整理



powered by amazon web services

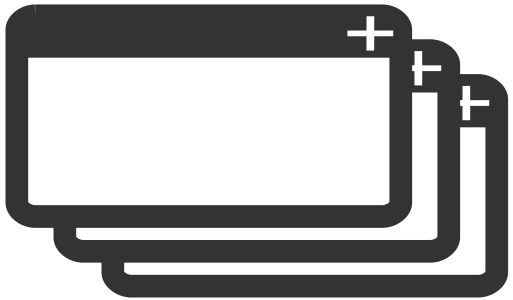
処理時間上限の解決

1. 処理の分割
2. 最適化
3. Java SpringFW利用の工夫

powered by
amazon
web services

処理の分割

画面のリスト作成
(Function名: Dispatcher)



HTML最適化
(Function名: Skeleton)

Javascriptコンパイル
(Function名: JS)

Lessコンパイル(CSS)
(Function名: Less)

powered by amazon web services

最適化

画面のリスト作成

Before:

1. RuntimeでJava classから情報収集
2. DBから設定データを読んでリスト作成

After:

1. Jar作成時に情報収集
2. DBから設定データを読んでリスト作成

powered by amazon web services

最適化

HTML最適化

Before:

HTML、JS、CSSのコンパイル全て一連の処理

After:

処理を切り離し、責務を明確に

powered by amazon web services

最適化

Javascriptコンパイル

Before:
　GoogleClosureCompilerをSpringFW上で動かす

After:
　GoogleClosureCompiler単体で動作

powered by amazon web services

最適化

Lessコンパイル（CSS）

Before:

Java＆SpringFW上で動作

After:

純正Node.jsコンパイラを利用

http://lesscss.org/

powered by amazon web services

SpringFW利用の工夫

- 工夫が必要だった理由

  初期化（Beanの生成）コスト 10 ~ 30 sec.

  画面数9000　x 30sec = 270000 sec

  = 4500 min

  ≒ **$4.5** (メモリ1Gの場合)

powered by amazon web services

SpringFW利用の工夫

- コンストラクタで節約
  - コンストラクタは課金対象外！
    - ただし処理時間が短い場合のみ
    - Timeout時間は短い&Timeoutすると課金対象

powered by amazon web services

SpringFW利用の工夫

*実装イメージ

```
1:  public class Main implements RequestHandler<S3Event, Object> {
2:      public Main() {
3:          …
4:        // ここに何か処理を書く
5:          …
6:      }
7:
8:      @Override
9:      public Object handleRequest(S3Event input, Context context) {
10:         // 各リクエストの処理ではこのメソッドが呼ばれる
11:     }
12: }
13:
```

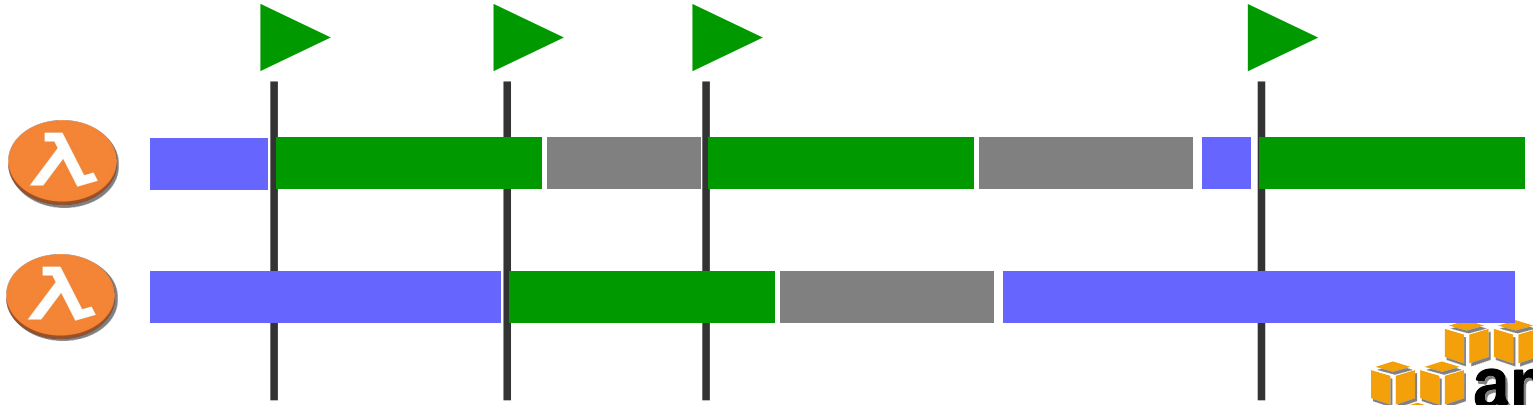
powered by amazon web services

SpringFW利用の工夫

- AWS Lambda Function のライフサイクル

  http://docs.aws.amazon.com/ja_jp/lambda/latest/dg/lambda-introduction.html

  - コールドスタート
  - コンテナ（≒JVM）の一定時間待機



powered by amazon web services

SpringFW利用の工夫

*実装イメージ

```
1:  public class Main implements RequestHandler<S3Event, Object> {
2:      private static ApplicationContext preheatContext;
3:
4:      @Override
5:      public Object handleRequest(S3Event input, Context context) {
6:          ...
7:              if (preheatContext != null && preheatContext.isActive()) {
8:                  logger.log("preheat spring context is already initialized.");
9:              } else {
10:                 preheatContext = new AnnotationConfigWebApplicationContext();
11:         …
12:     }
13: }
```

powered by amazon web services

Expand all Row Text

Filter events all 30s 5m 1h 6h 1d 1w custom

Message

2017-01-27 15:52:35

Start less compilation process: -8/85509/6.2e35ad61-e45d-11e6-a25c-e640338e2073.less.ac-autoposting-17.02.00.0000.20170109.1401.process

END RequestId: 10e715cb-e45d-11e6-8d44-cf4388d48e24

REPORT RequestId: 10e715cb-e45d-11e6-8d44-cf4388d48e24 Duration: 33111.13 ms Billed Duration: 33200 ms Memory Size: 1536 MB Max Memory Used: 235 MB

START RequestId: 089c21a2-e45d-11e6-91ed-69c4fa90ae92 Version: $LATEST

Receive a request: ObjectCreated:Put

Receive a request from Bucket: [redacted]

Receive a request with Parameter: [redacted]20170127155046-ac-autoposting-17.02.00.0000.20170109.1401/skeleton/934993374.skeleton.ac-autoposting-17.02.00.0000.20170109.1401.arguments

Preheat with application-code, application-version, tenant, landscape: ac-autoposting, 17.02.00.0000.20170109.1401, [redacted]

****************************** Start Preheat Job 20170127155046-ac-autoposting-17.02.00.0000.20170109.1401-934993374 ******************************

****************************** [1/3] Start To Prepare Execution Environment ******************************

preheat spring context is already initialized.

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

powered by amazon web services

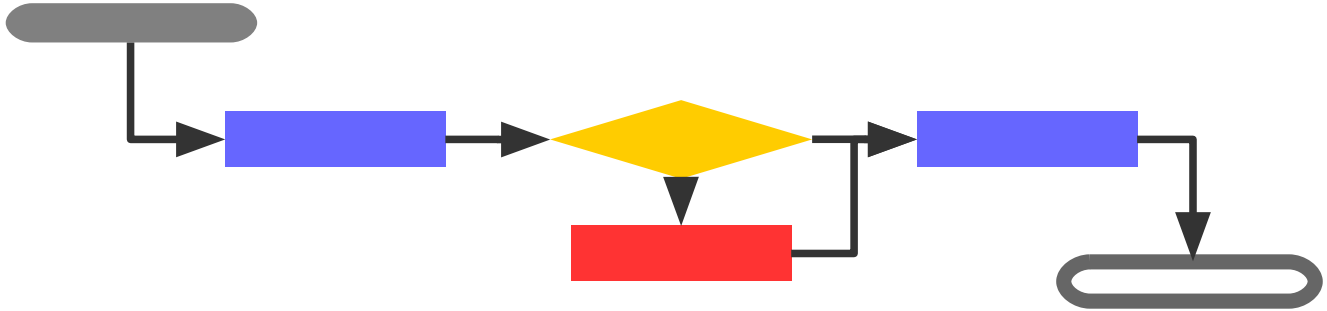
メモリ上限の解決

- 処理分割
- 3rd Party の見直し
  - Javascriptコンパイラ（Google closure compiler）にパフォーマンスチューニングが入っていた！(2016 / 09 / 11)

https://github.com/google/closure-compiler/wiki/Releases#september-11-2016-v20160911

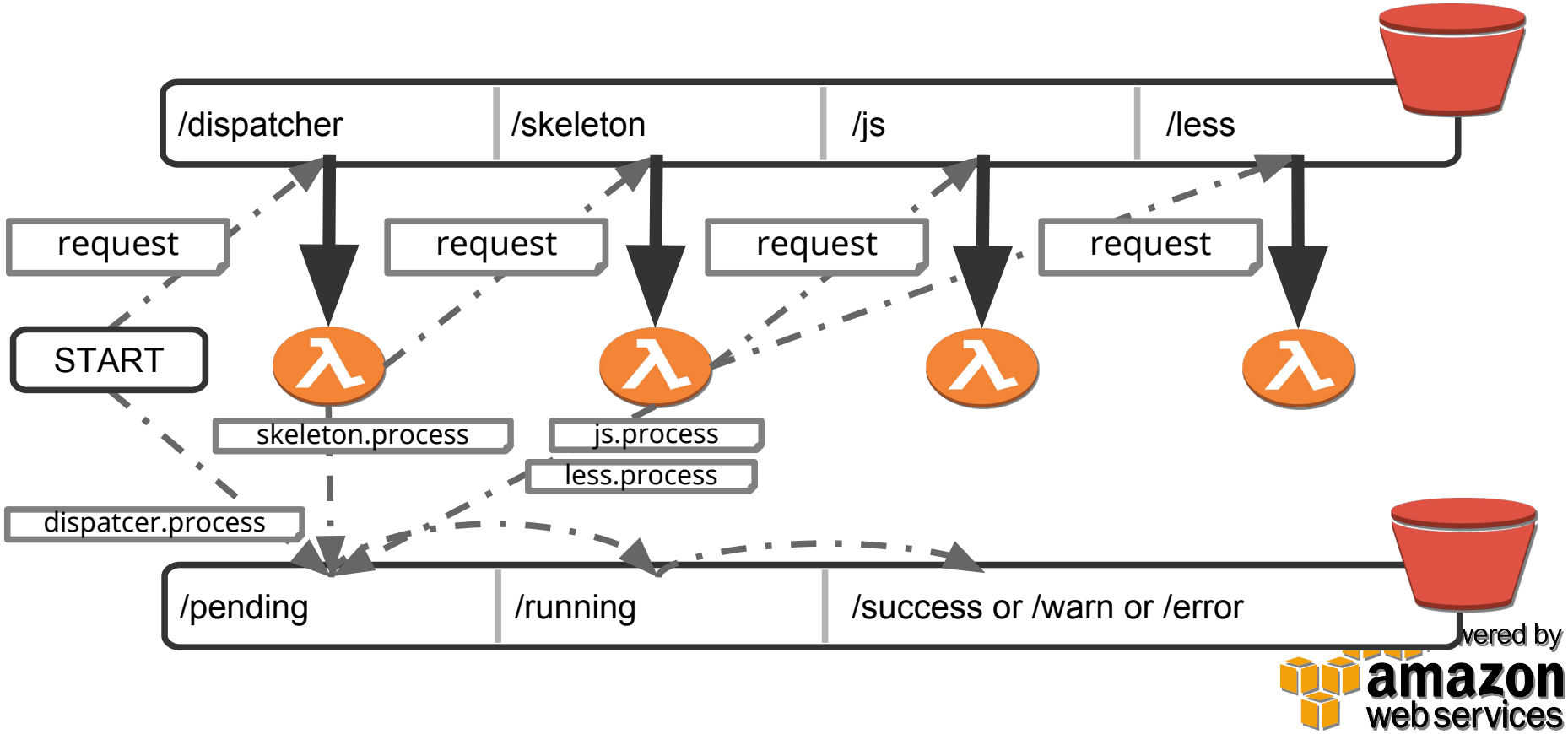
powered by amazon web services

バッチ処理フロー事例紹介

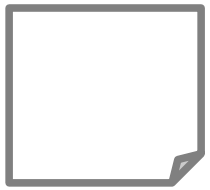


powered by amazon web services

処理フロー概要

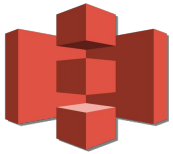

/dispatcher
/skeleton
/js
/less
request
request
request
request
START
skeleton.process
js.process
less.process
dispatcer.process
/pending
/running
/success or /warn or /error
vered by
amazon
web services

構成要素



**.request
イベント発火用

**.process
プロセス確認用



* For AWS Lambda

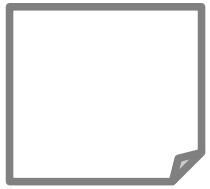
/{function_type}
/pending
/running
/success
/warn
/error

Dispatcher
Skeleton
JS
LESS

* For result

/result

powered by amazon web services



起動とプロセス管理

- 起動用ファイル: **.request
  - それぞれのLambda Function用に作成
  - suffixで処理を区別
  - S3にPutするとLambda Functionが実行される
  - 実行のEvidenceを残すために一度作ったら動かさない

- ステータス管理用ファイル: **.process
  - Pending -> Running -> Success
    Warn
    Error

*Pendingは一つ前のFunctionが作成

powered by amazon web services

[redacted]_develop_hr-payslip-less_event

Name

[redacted]_develop_hr-payslip-less_event

Events

- [ ] RRSObjectLost
- [x] Put
- [ ] Post
- [ ] Copy
- [ ] Complete Multipart Upload
- [ ] Delete
- [ ] Delete Marker Created
- [ ] ObjectCreate (All)
- [ ] ObjectDelete (All)

Prefix

[redacted]

Suffix

less.hr-payslip-17.02.00.0000.20170109.1401.request

Send to

Lambda Function

Lambda

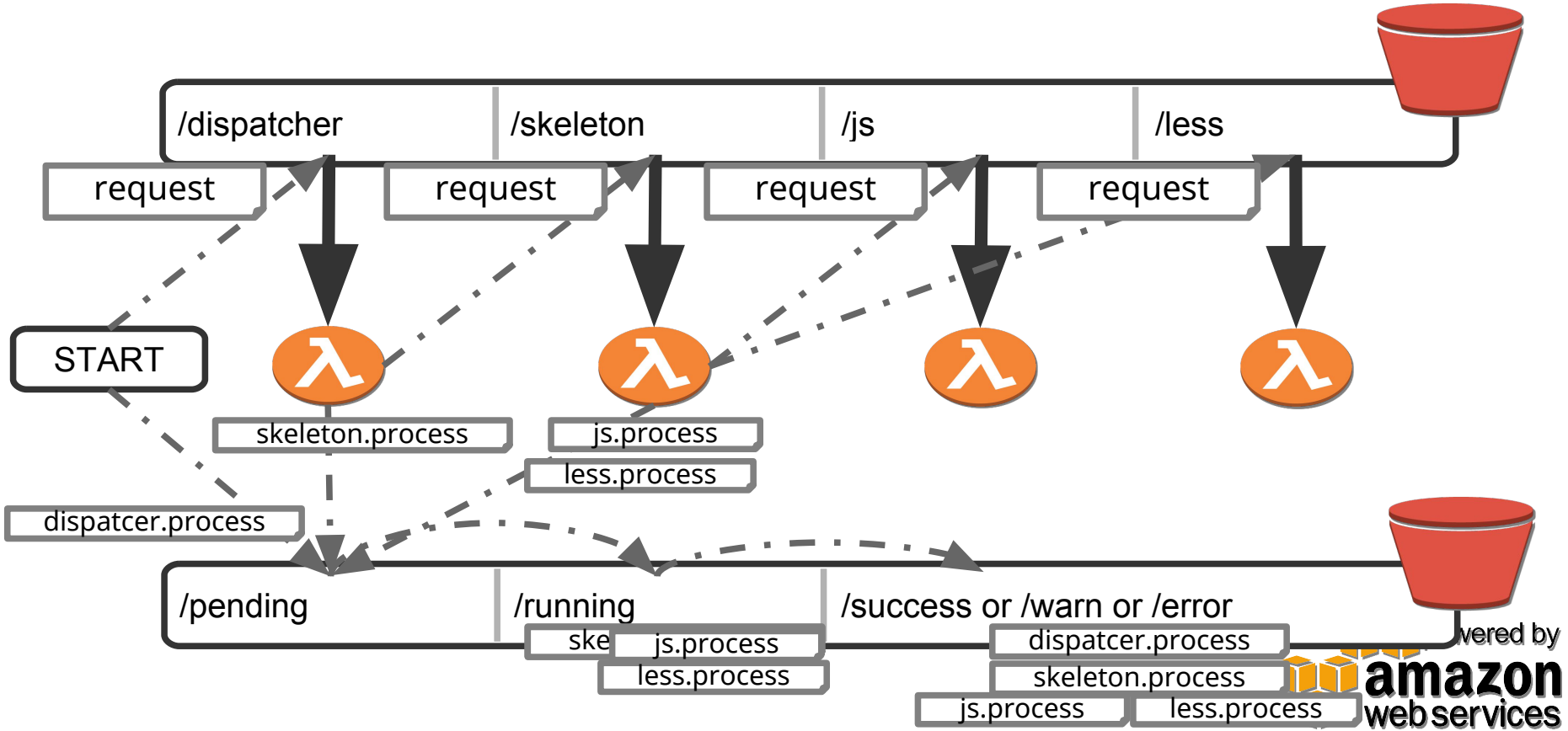
Add Lambda function ARN

Lambda function ARN

arn:aws:lambda:[redacted]:function:gtln_develop_hr-payslip-less

powered by amazon web services

処理フロー概要


/dispatcher
/skeleton
/js
/less
request
request
request
request
START
skeleton.process
js.process
less.process
dispatcer.process
/pending
/running
/success or /warn or /error
ske
js.process
less.process
dispatcer.process
skeleton.process
js.process
less.process
vered by
amazon
web services

進捗確認

- S3を確認
  - ディレクトリ名がステータス
  - ListObject APIで取得

```
$ echo -e "pending/\nrunning/\nsuccess/\nwarn/\nerror/" | \
xargs -I{} bash -c  "echo {};aws s3 ls s3://${bucket}/${tenant}/20170405133635-collabo-latest/{} | wc -l"

pending/   0
running/   0
success/   1330
warn/   80
error/   44
```

powered by amazon web services

Troubleshooting事例

powered by amazon web services

Troubleshooting事例

問題:

速度が思ったより速くない

- CPU性能はメモリに比例　128MB - 1536MB

解決策:

- メモリ上限UP!
- 1.5Gだと2コア(2/3コア)利用可能
- コスト増ほぼなし

powered by amazon web services

Troubleshooting事例

問題点:

上限変更後、並列数が伸びない

- 何度も問い合わせ・・・

解決策:

- VPCの設定を確認(subnet)
- 必要ないなら設定しない。

powered by amazon web services

Troubleshooting事例

問題:

Lessコンパイルが起動後すぐにエラーになる場合がある

- 唯一、ランタイムがNode.js
- S3への大量Put＆結果整合性

powered by amazon web services

```
START RequestId: 2199d19f-e301-11e6-88b1-c137133b5dfc Version: $LATEST
2017-01-25T13:21:08.027Z   2199d19f-e301-11e6-88b1-c137133b5dfc    Reading options from event:
{
    "Records": [
        {
            "eventVersion": "2.0",
            "eventSource": "aws:s3",
            "awsRegion": "ap-northeast-1",
            "eventTime": "2017-01-25T13:21:07.907Z",
            "eventName": "ObjectCreated:Put",
            "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "***_develop_ac-autoposting-less_event",
                "bucket": {
                    "name": "**********************",
                    "ownerIdentity": {
                        "principalId": "A9N1TG2KH40T"
                    },
                    "arn": "arn:aws:s3:::**********************"
                },
                "object": {
                    "key":
"*****/20170125131913-ac-autoposting-17.02.00.0000.20170125.0201/less/-1434027423.14185431-e301-11e6-8a92-222fd625ffe8
```

$ aws s3 ls s3://${BUCKET}/${TENANT}/20170125131913-ac-autoposting-17.02.00.0000.20170125.0201/less/-1434027423.14185431-e301-11e6-8a92-222fd625ffe8.less.ac-autoposting-17.02.00.0000.20170125.0201.request

2017-01-25 22:21:08 ＊JSTなので9時間差

差51569 -1434027423.14185431-e301-11e6-8a92-222fd625ffe8.less.ac-autoposting-17.02.00.0000.20170125.0201.request

wered by amazon web services

Troubleshooting事例

問題:

Lessコンパイルが起動後すぐにエラーになる場合がある

- 唯一、ランタイムがNode.js
- S3への大量Put＆結果整合性

解決策:

- S3にオブジェクトがあるかはじめに確認
  - retry 3回だがわかっているものはちゃんと処理

powered by amazon web services

Troubleshooting事例

問題:

S3イベントが登録されない！

-> 登録したはずが、消えている

- **全部で100Function**
- Ansible でのS3イベント登録は1件ずつ

解決策:

- 一度に全てのを登録！
  - Ansibleを改良(python)

powered by amazon web services

Troubleshooting事例

問題点:

想定外の金額。合計金額を確認すると大変なことに

- CloudWatchのログから簡単に金額計算

```
REPORT RequestId: b33f5ffc-28a1-11e7-88fd-bbda627ccea3
Duration: 4528.02 ms    Billed Duration: 4600 ms Memory Size: 1536 MB
    Max Memory Used: 275 MB
```

解決策:

- ローカルファイルを都度cleanする。
  - /tmp/hoge -> s3/${bucket}/hogeにアップ
  - /tmp/hogeには前回の処理のファイルがある

powered by amazon web services

出荷

- 出荷
  - Mavenでassemble
  - AWS Lambda の Functionは zip形式
    - 25製品それぞれに最適化されたzip

<ライブラリ>

[Java]
dispatcher.jar
skeleton.jar
js.jar

[Node.js]
less.zip

<AWS Lambda Function>

**-dispatcher.zip
**-skeleton.zip
**-js.zip

**-less.zip

powered by amazon web services

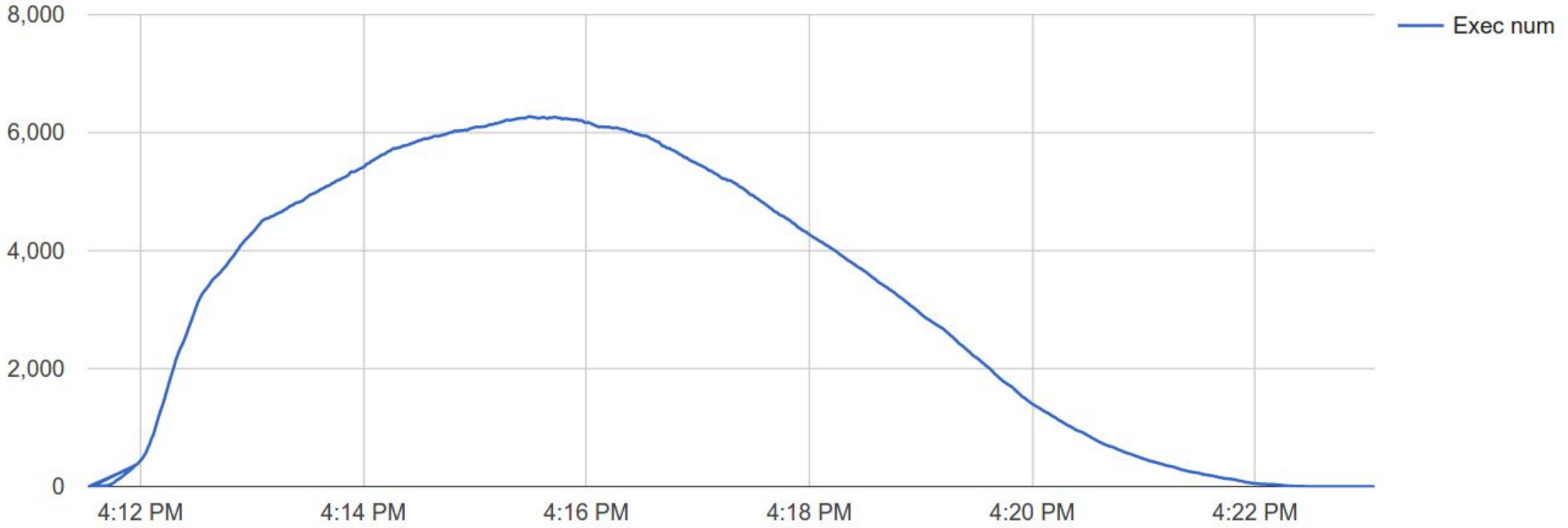
運用（初期設定）

- お客様毎に別VPC
  - VPC1環境にLambdaFunctionを作成
  - 1VPC1バケット

powered by amazon web services

効果測定

powered by amazon web services



しかし

powered by amazon web services

問題

- DBが高負荷
  - 事前処理のみ実行
    - 1回目 成功
    - 2回目 2/8000 高負荷によるエラー
  - APサーバー実行時
    - 1回目 成功
    - 2回目 10/8000 高負荷によるエラー

powered by amazon web services

今後の課題

powered by amazon web services