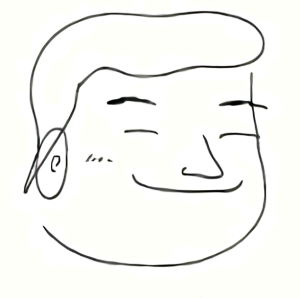
fujiwara-ware OSSをひたすら紹介する

Ya8 2024 - 2024/03/15

@fujiwara 藤原俊一郎

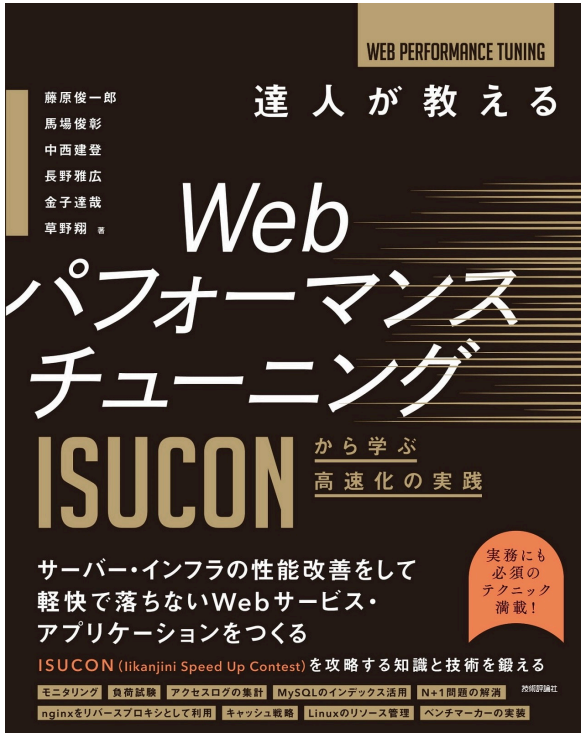
自己紹介



面白法人カヤック

@fujiwara (X/Twitter, GitHub, Bluesky)

面白法人カヤック SREチーム
ISUCON 🥇 優勝 4回
ISUCON 運営(出題) 4回



あらすじ

"最近の趣味はマネージドサービスの隙間を埋める隙間家具のようなツールをGoで作ってOSSにすること" と公言してるとおり、OSSをいくつも作っています。

それらは主に Go で書かれた、AWS のマネージドサービスを便利に使うためのものなのですが、作ったはいいもののblogすら書いていなかったりして、世間に存在を知られていないものもあるなと気が付きました。

そこで今回は、自作OSSをひたすら紹介しようと思います。もしかしたらその中から、あなたのお仕事に便利に使える道具がなにか見つかるかもしれません。

お品書き

過去1年以内になんらかの手を入れたものを紹介します

cfft

CloudFront FunctionsのテストとCloudFront KVSの操作を便利にするやつ [New!]

ecspresso

Amazon ECSデプロイツール。多分一番有名?

lambroll

AWS Lambdaデプロイツール
つい最近 v1.0 を出しました。新機能もあるよ

ecsta

Amazon ECSのタスク操作を便利にできるツール
ecspresso からの派生品です

tracer

ECSタスクの一生を全部まとめてみるやつ

ridge

LambdaのFunctionURL/ALB/API GatewayのリクエストをGoの標準httpモジュールで書けるやつ

lamblocal

Goで、LambdaでもCLIでも動くような1つのバイナリを作るのに便利なユーティリティ

tfstate-lookup

Terraform tfstateを取得してparseして中身を見るCLI/ライブラリ

go-amzn-oidc

Amazon ALB OIDC認証で渡されるJWTを検証するやつ

ecrm

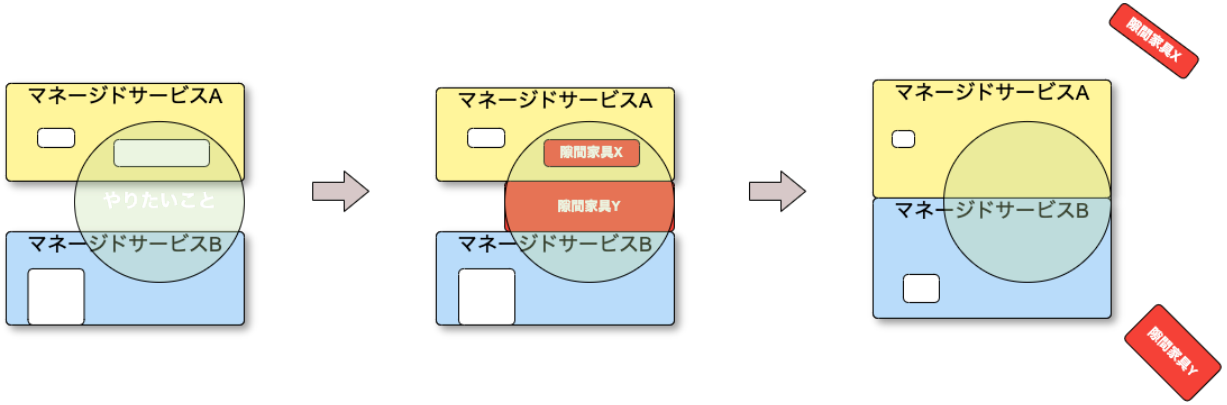
Amazon ECRを安全に掃除してくれるやつ

riex

買っている/もうすぐ期限が切れそうなReserved Instanceを表示してくれるやつ

(前提)なぜ大量の隙間家具OSSをGoで書いているのか

隙間家具 = マネージドサービスの隙間を埋めて便利にするソフトウェア



仕事でちょっとした不満、不便があるものを解消したい (結局コードは書く)
ツールはオープンにしておかないと腐りがち (コピペ→魔改造)
むしろOSSのほうがよい設計になる (ツールの責務/責任分解点が明確に)
GoもマネージドサービスのAPIも互換性を重視 = メンテなしでも動き続ける
要するに**実利があるのでやっている**

cfft ⭐18

https://github.com/fujiwara/cfft
CloudFront Functions(CFF)のテストを便利にするためのツール
解説記事: https://techblog.kayac.com/cfft-test-cloudfront-functions

解決したかった不満

CFFのJSランタイムは公開されていないので、手元でのテストができない
マネコン/CLI/APIではできるが、面倒くさいので簡単にできるようにした

推しポイント

CF KVSといい感じに統合(IDをハードコードしなくていい)
複数の関数を結合してひとつの関数にビルドできる(CFFは1つの関数しか呼べない)
Terraformといい感じに連携できる

ecspresso ⭐715

https://github.com/kayac/ecspresso
自分の代表作といっていいやつ
Amazon ECSデプロイツール

解決したかった不満

書いてみたら300行で実用になったのでやった(本音 以下は後付けの理由です)

ECSで動くアプリとそれ以外の周辺リソースはライフサイクル(更新頻度)が違う
なんなら担当チームも違ったりする(アプリ / インフラみたいな)
アプリケーションのデプロイ(ECSのタスク定義とサービス)だけちゃんとやりたい

hakoも検討したけど、ECS専用にしたほうが理解しやすそう

ecspresso 推しポイント

ECSサービス/タスク定義以外の余計なリソースをなるべく作らない/管理しない

既存のECSサービスを一発でコード管理できる(`ecspresso init`)

デプロイ以外にもECS運用でかゆいところに手が届く便利機能がいっぱい
(`diff, verify, rollback, exec, refresh, scale, run, status...`)

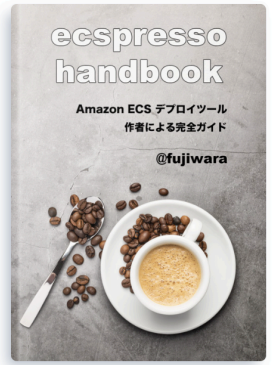
国内採用実績が豊富(はてな、ニンテンドーシステムズなど数十社以上)

Terraform tfstate参照機能が便利

ecspresso handbook

https://zenn.dev/fujiwara/books/ecspresso-handbook-v2

Zenn



ecspresso handbook v2対応版

fujiwara

Amazon ECS デプロイツール ecspresso の作者による解説本です。チュートリアル、基本的な使いかた、応用的な使いかた、設計思想と実装、全コマンドのリファレンスを掲載しています。

本書は[ecspresso handbook v1対応版](https://zenn.dev/fujiwara/books/ecspresso-handbook)を元に、2022年12月14日にリリースされたecspresso v2への対応と構成変更、新章の追加を行ったものです。

v1対応版とは多くの内容が重複しているため、購入の際にはご注意ください。

2023-12-21 v2.3.0対応版に更新しました。

今すぐ読む

GitHubで開く

公開	2022/12/23
本文更新	2023/12/21
文章量	約129,845字
価格	500円

106 ポスト

12万字(紙だったら100ページ超)で 500円、お得です

lambroll ⭐303

https://github.com/fujiwara/lambroll
AWS Lambdaデプロイツール ecspressoのLambda版だと思ってもらえれば
2024年1月に v1.0 を出しました

解決したかった不満

Apex (github.com/apex/apex) がアーカイブされてしまったので代替がほしかった

SAMもServelessも余計なことする(主観)し、ついカッとして書いたらできた

lambroll 推しポイント

Lambda関数(+FunctionURLs)以外の余計なリソースを作らない/管理しない

Zip / コンテナImage両対応

既存のLambda関数を一発でコード管理できる(`lambroll init`)

`lambroll logs --follow` でログが見られて便利

Terraform tfstate参照もできる

ecsta ⭐45

https://github.com/fujiwara/ecsta
Amazon ECSのタスク操作を便利にできるツール

ecspresso exec(ECS Exec)、tasks(タスク一覧を見たり停止したり)を単体ツールに
(ecspresso v2からはecstaが本体)

解決したかった不満

元々ecspressoにあった機能だが、責務的に単体タスクはecspressoの範囲外だなと
思ったので切り出した

ecsta 推しポイント

ecspresso管理でもない任意のECSタスクを操作できる
マネコンいらないぐらい便利(主観)

`exec` : ECS Execする(ログイン、コマンド実行)
`portforward` : 指定したタスクにポートフォワード(手元からRDSに繋いだりも)
`trace` : `tracer` 実行(後述)
`logs` : ログをtailする

tracer ⭐86

https://github.com/fujiwara/tracer
あるECSタスクの一生(イベント、CloudWatch Logs)を時系列で全部見る

解説記事 https://techblog.kayac.com/ecs-task-tracer

解決したかった課題

ECSのタスク、起動しなかったり起動できてもすぐ落ちたりする
原因を調べるのにいろんなところを見て回るのがつらすぎる

一目で全部のログが見えるツールないの? → ぱっと見なかったので書いたらできた
(金曜の夜に酒飲んでたら一晩で最初のバージョンはできた)

tracer 推しポイント

あるタスクが起動してから終了するまで

なにが起きているのかが時系列になって出力されるので一目瞭然

```
$ tracer default 9f654c76cde14c7c85cf54dce087658a
2021-12-04T09:03:01.504+09:00   TASK    Created
2021-12-04T09:03:05.375+09:00   TASK    Connected
2021-12-04T09:03:17.878+09:00   TASK    Pull started
2021-12-04T09:03:25.285+09:00   TASK    Pull stopped
2021-12-04T09:03:25.927+09:00   CONTAINER:nginx /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
2021-12-04T09:03:25.928+09:00   CONTAINER:nginx /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
2021-12-04T09:03:25.929+09:00   CONTAINER:nginx /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
2021-12-04T09:03:25.935+09:00   CONTAINER:nginx 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
2021-12-04T09:03:25.945+09:00   CONTAINER:nginx 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
2021-12-04T09:03:25.945+09:00   CONTAINER:nginx /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
2021-12-04T09:03:25.951+09:00   CONTAINER:nginx /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
2021-12-04T09:03:25.951+09:00   CONTAINER:nginx /docker-entrypoint.sh: Configuration complete; ready for start up
2021-12-04T09:03:25.954+09:00   CONTAINER:nginx nginx: invalid option: "x"
2021-12-04T09:03:25.990+09:00   TASK    Execution stopped
2021-12-04T09:03:26.006+09:00   TASK    Started
2021-12-04T09:03:36.060+09:00   TASK    Stopping
2021-12-04T09:03:36.060+09:00   TASK    StoppedReason:Essential container in task exited
2021-12-04T09:03:36.060+09:00   TASK    StoppedCode:EssentialContainerExited
2021-12-04T09:03:49.533+09:00   TASK    Stopped
2021-12-04T09:03:49.533+09:00   CONTAINER:nginx STOPPED (exit code: 1)
```

tracer 推しポイント

ECSタスクにトラブルがあったときに**マネコンをいったりきたりしなくてよい**

ライブラリとしても呼べる(ecsta / ecspresso に組み込み済み)

`tracer` バイナリをLambdaのZipに `bootstrap` という名前で入れておくと

EventBridgeのECSタスク停止イベントを元に実行するLambda関数としても動く

ridge ⭐51

https://github.com/fujiwara/ridge

LambdaのFunctionURL/ALB/API GatewayをGoの標準httpモジュールで書けるやつ

```
import (
  "net/http"
  "github.com/fujiwara/ridge"
)
func main() {
  var mux = http.NewServeMux()
  mux.HandleFunc("/", handleRoot)
  ridge.Run(":8080", "/", mux)
}
func handleRoot(w http.ResponseWriter, r *http.Request) {
  w.Header().Set("Content-Type", "text/plain")
  fmt.Fprintln(w, "Hello World")
  fmt.Fprintln(w, r.URL)
}
```

解決したかった不満

Lambda API Gateway Integration(2016年当時)、可能性は感じるけど面倒くさいな

net/httpとの相互変換を書いたらLambda/API Gateway固有の事情を忘れられるのでは?

ridge 推しポイント

普通にnet/httpのWebアプリを書いたバイナリがなにもしないでLambdaで動く

もしECSやEC2で動かしたくなっても再ビルド不要でバイナリがそのまま動く

FunctionURLがでたので実用性500%UP(API Gateway比、作者調べ)
lambroll v1でfunction URLも一発デプロイできるようになったので更に便利に

(古い)解説スライド: https://speakerdeck.com/fujiwara3/go-on-aws

lamblocal ⭐9

https://github.com/fujiwara/lamblocal
Goで、LambdaでもCLIでも動く1つのバイナリを作るのに便利なユーティリティ
解説記事: https://sfujiwara.hatenablog.com/entry/lamblocal

解決したかった不満

Lambdaで書くのはちょっとしたコードなのに、
動作確認やデバッグをいちいちLambdaにデプロイしてやるのは面倒

ちょっとしたコード、普通に手元でも実行したいことが多い

何も考えずにLambdaをGoで書くと、設定値の扱いがカオスになりがち
(いろんなところで `os.Getenv()` してたりする)

lamblocal 推しポイント

```go
func handler(ctx context.Context, payload any) (string, error) {
  // 何かするコード
  return "ok", nil
}

func main() {
  lamblocal.Run(context.TODO(), handler)
}
```

Lambda上ではLambda関数として、それ以外では普通のCLIコマンドとして動く

CLIの場合 `payload` は標準入力からJSONで渡せる

lamblocal 推しポイント

環境変数を扱えるflag parserと組み合わせるときれいに書ける例

```
import "github.com/alecthomas/kong"
type CLI struct {
  Foo string `help:"Foo." default:"foo" env:"FOO"`
}
func (c *CLI) Handler(ctx context.Context, _ any) (string, error) {
  // c.Foo はデフォルト値、環境変数、コマンドライン引数から設定された状態になっている
  return "OK", nil
}
func main() {
  var c CLI
  kong.Parse(&c)
  lamblocal.Run(context.TODO(), c.Handler)
}
```

tfstate-lookup ⭐73

https://github.com/fujiwara/tfstate-lookup

Terraform tfstateを取得してparseして中身を見るCLI/ライブラリ

ecspresso, lambrollのtfstate参照はこれを使っている

`terraform state show` と同じようなことができる、値の中も掘れる

```
$ tfstate-lookup aws_vpc.main
{
  "arn": "arn:aws:ec2:ap-northeast-1:123456789012:vpc/vpc-xxxxxxxx",
  "assign_generated_ipv6_cidr_block": false,
  "cidr_block": "10.0.0.0/16",
  "default_network_acl_id": "acl-yyyyyyy",
  ...

$ tfstate-lookup aws_vpc.main.cidr_block
10.0.0.0/16
```

解決したかった不満

ecspressoの設定ファイルに各種リソース(Subnet,SecurityGroupなど)のIDを
ハードコードしたくない

aws cliでIDを取って環境変数にセットして `{{ env "F00" }}` で参照、を繰り返していたら
嫌になった

terraformのコードをimportして使う手はあるか?と一瞬思ったが、やらなくて正解
(その後terraformのコードはinternal化された)

tfstate-lookup 推しポイント

`terraform state show` より速い(400リソースあるS3のstateで300%以上)

```
$ terraform state show aws_vpc.main
  1.65s user 0.34s system 95% cpu 2.073 total
$ time tfstate-lookup aws_vpc.main
  0.04s user 0.03s system 11% cpu 0.649 total
```

File / S3 / GCS / AzureRM / TerraformCloud のtfstateを直接参照できる(terraform不要)

インタラクティブモード `tfstate-lookup -i` が便利

hemlfile に依存されている (正確には `helmfile/vals` から)

ecschedule でも使われている

go-amzn-oidc ⭐3

https://github.com/fujiwara/go-amzn-oidc

Amazon ALB OIDC認証で渡されるJWTを検証するやつ

ALB(Application Load Balancer)のOIDC認証機能を使うと
認証を通ったあと `x-amzn-oidc-data` というヘッダでJWTが渡される(要署名検証)

解決したかった不満

署名検証は公開鍵を以下のURLから取得する必要がある

`https://public-keys.auth.elb.{region}.amazonaws.com/{jwtに入っているkeyid}`

リクエストごとに取得するのは遅いのでcacheとか考えると都度書くのは面倒

それをやってくれるライブラリ

```
import "github.com/fujiwara/go-amzn-oidc/validator"
claims, _ := validator.Validate(r.Header.Get("x-amzn-oidc-data"))
fmt.Fprintln(w, claims.Email())
```

go-amzn-oidc 推しポイント

実は `x-amzn-oidc-data` は**ValidなJWTではない**
JWTではbase64のpadding(末尾の `=`)を付けないのが仕様だがALBは `=` を付けてくる
JWT署名が `=` 込みで計算されているため、単純に無視もできない

世間のJWTライブラリを普通に使うと一生検証が通らないのでつらい

`golang-jwt/jwt/v4` の `DecodePaddingAllowed=true` を設定してあえて通している

go-amzn-oidc 推しポイント

単体のサーバーとして起動するとnginxの `auth_request` と組み合わせて使える

アプリケーションがGo以外の言語の場合

JWT検証済のclaims.Emailをヘッダで引き渡してなんとかできる

```
location = /oidc_validate {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header X-Amzn-OIDC-Data $http_x_amzn_oidc_data;
        proxy_set_header Content-Length "";
        proxy_pass_request_body off;
        internal;
}

location / {
        auth_request /oidc_validate;
        auth_request_set $email $upstream_http_x_auth_request_email;
        proxy_set_header X-Email $email;
        # ...
}
```

ecrm ⭐51

https://github.com/fujiwara/ecrm
Amazon ECRを安全に掃除してくれるやつ

解決したかった不満

ECR(Elastic Container Registry)のライフサイクルルールは危険
ECSで今現在使っているimageも気にせずに「ルールに従って」削除する
(ので何回か事故った)

ECS/Lambdaで今(+指定した世代)使用中のimageを避けてそれ以外を削除するツール

ecrm 推しポイント

解説記事 https://techblog.kayac.com/ecrm-oss

`ecrm init` でアカウント内のリソースを元にいい感じのconfigを自動生成

```
clusters:
  - name_pattern: prod-*
task_definitions:
  - name_pattern: prod-*
    keep_count: 5
lambda_functions:
  - name_pattern: prod-*
    keep_count: 5
    keep_aliase: true
repositories:
  - name_pattern: prod/*
    expires: 30d
    keep_count: 5
    keep_tag_patterns:
      - latest
```

ecrm 推しポイント

`ecrm plan` で削除される予定のimageの数(サイズ)を表示してくれる

```
        REPOSITORY        |     TOTAL      |     EXPIRED      |     KEEP
--------------------------+----------------+------------------+--------------
       dev/app            | 732 (594 GB)   | -707 (574 GB)    | 25 (21 GB)
       dev/nginx          | 720 (28 GB)    | -697 (27 GB)     | 23 (875 MB)
       prod/app           | 97 (80 GB)     | -87 (72 GB)      | 10 (8.4 GB)
       prod/nginx         | 95 (3.7 GB)    | -85 (3.3 GB)     | 10 (381 MB)
```

注意: 削減サイズはECR上の重複layerを考慮できないのでかなり過大に出がち

riex ⭐4

https://github.com/fujiwara/riex

買っている/もうすぐ期限が切れそうなRI(Reserved Instance)を表示してくれるやつ

解決したかった不満

RIが期限切れたのに気が付かなくて 💸💸💸

riex 推しポイント

markdownで出せるのでGitHub Actionsでまわしてprにしたらいい感じになる

```
$ riex 30 --format markdown
```

service	name	description	instance_type	count	start_time	
RDS	prod-ce-8x-2	aurora-mysql	db.r6g.8xlarge	1	2022-10-14T08:09:30Z	2
Redshift	c36868e7-5421-41d0-ab87-841a0d162d1f		ra3.xlplus	1	2022-12-21T08:02:18Z	2
	ri-2022-08					

まとめ

仕事でちょっとした不満、不便があるものを解消したい
→ コードを書かないで解決できれば一番だけど、書いて解決してよいこともある

書いたらコードはオープンにしておかないと腐ります
→ 実際何度も腐った

むしろOSSにする前提のほうがよい設計になる
→ 固有の要件とそうでないものを切り分ける訓練になります

いざという時にコードで解決するためには、普段から書かないといけない
→ 普段やってないことはいきなりうまくはできない (たとえばISUCON)

つまり、**どんどんやっていきましょう**
うまくいかなかったら捨てて次に生かしましょう
捨てるためには、密結合しないできれいに外せる設計も考えましょう