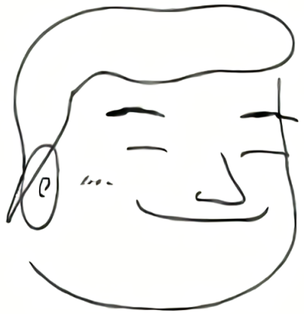
Amazon ECSで好きなだけ検証環境を起動できるOSSの設計・実装・運用

YAPC::Hiroshima 2024

@fujiwara 藤原俊一郎

自己紹介



面白法人カヤック

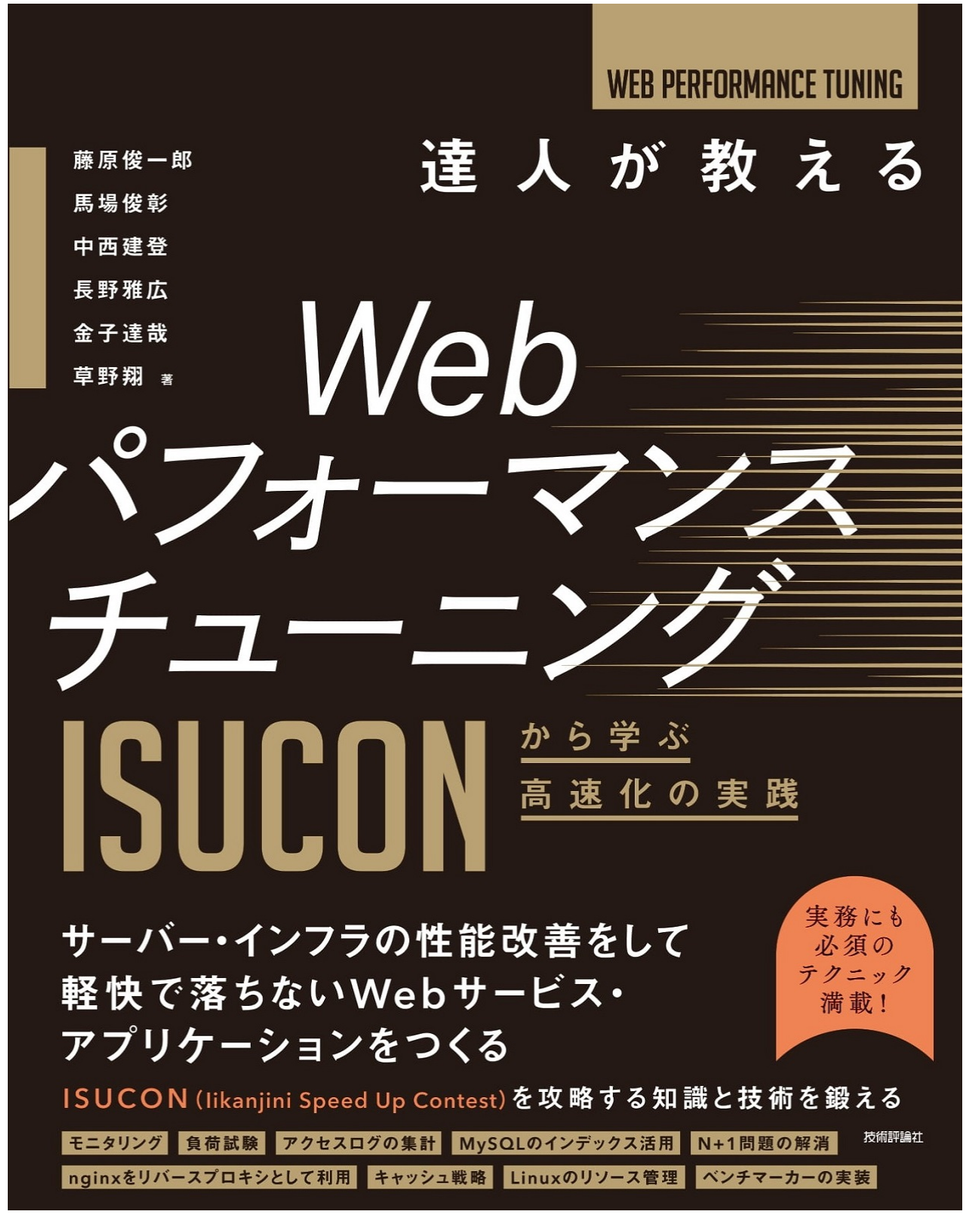
@fujiwara 面白法人カヤックSREチーム

ISUCON 1,2,5,11 🥇 優勝4回

ISUCON 3,8,12,13 運営(出題) 4回

github.com/kayac/**ecspresso**
Amazon ECS デプロイツール

github.com/fujiwara/**lambroll**
AWS Lambda デプロイツール



Agenda

ブランチ別開発・検証環境の実現方法と課題

検証環境を容易に起動できるOSS mirage-ecs

mirage-ecsの設計と実装

mirage-ecsの実践的な運用

ブランチ別開発・検証環境の実現方法と課題

「ブランチ別開発・検証環境」

(ここでの定義)

「Webサービスのサーバーを任意の(リポジトリのブランチの)状態で起動して、独立したURLで外部からアクセスできるようにしたもの」

「ブランチ別開発・検証環境」が必要な場面

エンジニアはローカルで開発環境を起動できる、が
それ以外の人も開発中のサーバーにアクセスしたいことは多い

チーム外の人に開発中の状態を見てもらいたい

- お客さんとか関係者

複数人で同じ環境に同時にアクセスしたい

- ゲームだとチームチェックという文化がある
- 複数人でアクセスしないと成立しないアプリケーション

「ブランチ別開発・検証環境」が必要な場面

エンジニア以外でも自分専用のサーバーがほしい

- テスト、QAのため
- デザイナーが画像やアセットを実機確認するため
- ディレクターがマスターデータやお知らせなどを確認するため

環境の起動、停止は**誰でもできる**ようになっていてほしい

「XXX環境を起動お願いします！」「更新お願いします！」...
エンジニアがいちいち対応していたら大変

実現方法(1)

「事前に複数環境を作成して使い回す」

あらかじめ、環境を複数用意しておく
環境は任意のブランチに切り替えて起動できるようにしておく

使いたい人が空いている環境を
任意のブランチで起動して使う

問題点

- 誰がどこを使っているかを管理する必要
- 使い終わったら速やかに開放しないと他の人が使えない
- かといって全員分用意すると高コスト

実現方法(2)

「必要な時にIaCで全部作る」

使いたいタイミングで必要なサーバーリソースを一式まるっと作成

例: AWSでALB + ECS + RDS + ElastiCache

ALB, ECSサービス/タスク, RDS, ElastCache を作成できるコード(TerraformとかCFnとかCDKとか)を作っておいてバーン 🚀

「必要な時にIaCで全部作る」 方法の問題点

作るのに時間が掛かる(やつがある)

- 新規のRDSやElastiCacheを起動すると20分〜とか

一式丸ごと用意するととにかくお金が掛かる

- 消し忘れると... 💸

共用リソースと動的な作成/削除とIaCの相性が微妙(なことがある)

- コスト削減のために一部のリソース(LBなど)を共用すると起きがち
- 環境Aの作成とBの削除は平行でできない、とか

方法(1)(2)の問題点

あらかじめ用意しておくと取り合いに(**管理コスト**)

足りないと待ち時間が無駄に(**時間**)

全員分を常に用意するとお金が掛かる(**インフラコスト**)

起動と停止に時間が掛かると待ち時間が無駄(**時間**)

時間が掛かると手放したくなくなる(不足・放置→**インフラコスト**)

いつでも必要なだけ、高速に起動できて低コストな仕組みがほしい

いつでも必要なだけ
高速に起動できて
しかも低コスト
を実現する

mirage-ecs

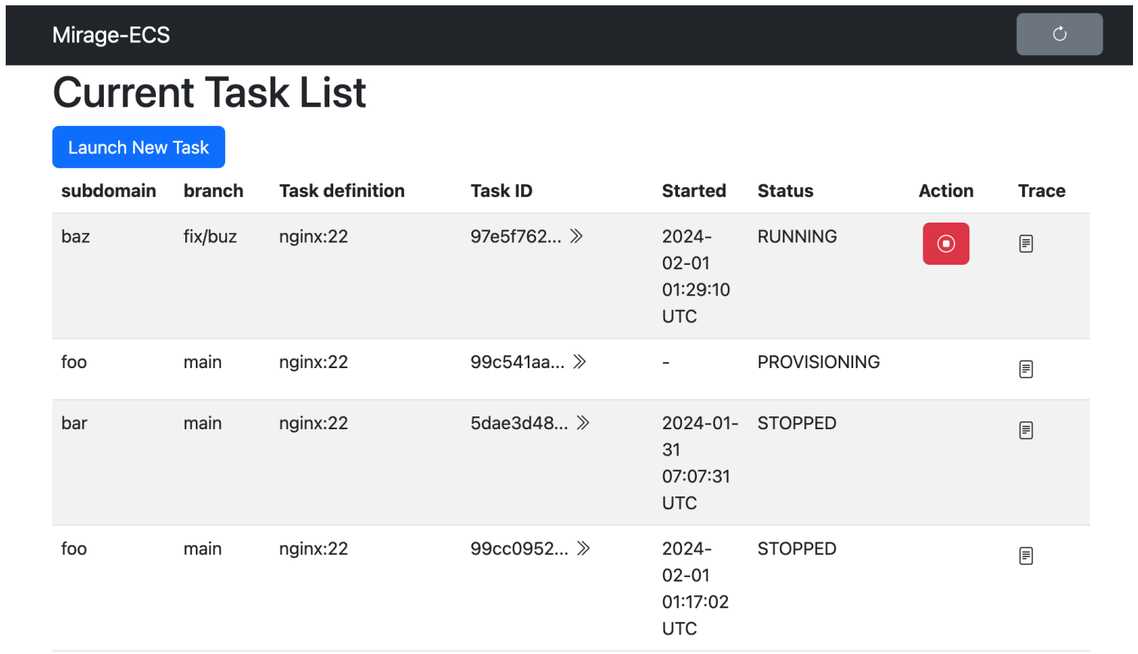
「ブランチ別開発・検証環境」を高速に起動できる
低コストで実現するソフトウェア(OSS)

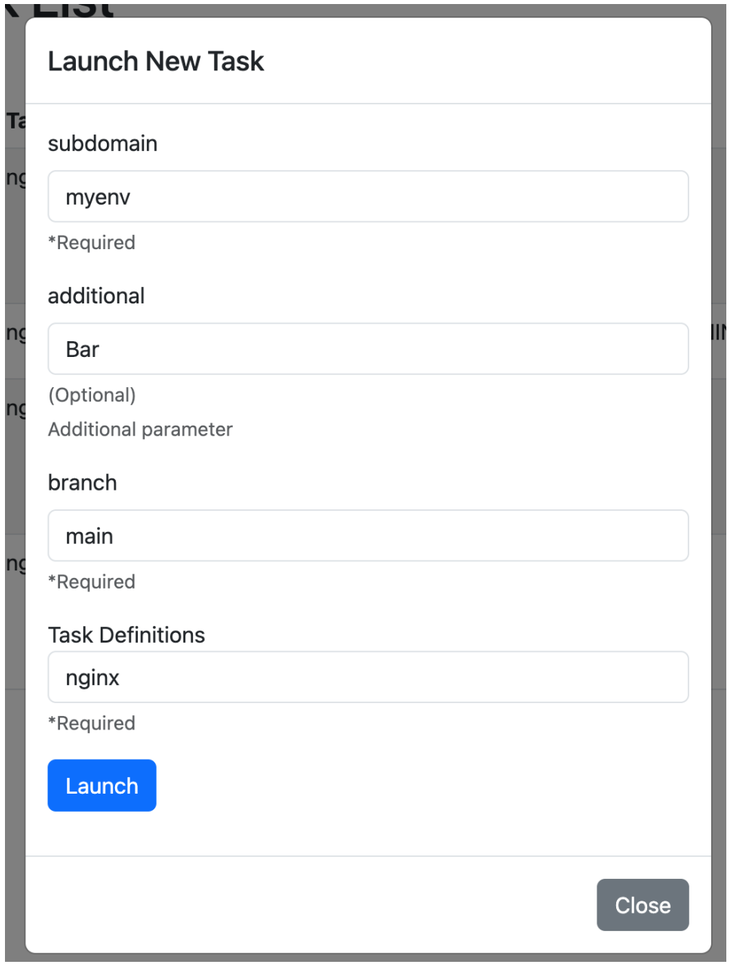
github.com/acidlemon/mirage-ecs

mirage-ecs is a reverse proxy for ECS tasks and a manager for the tasks.

ECSタスクのマネージャー兼リバースプロキシ

Demo


Mirage-ECS
Current Task List
Launch New Task
subdomain branch Task definition Task ID Started Status Action Trace
baz fix/buz nginx:22 97e5f762... 2024-02-01 01:29:10 UTC RUNNING
foo main nginx:22 99c541aa... - PROVISIONING
bar main nginx:22 5dae3d48... 2024-01-31 07:07:31 UTC STOPPED
foo main nginx:22 99cc0952... 2024-02-01 01:17:02 UTC STOPPED


Launch New Task
subdomain
myenv
*Required
additional
Bar
(Optional)
Additional parameter
branch
main
*Required
Task Definitions
nginx
*Required
Launch
Close

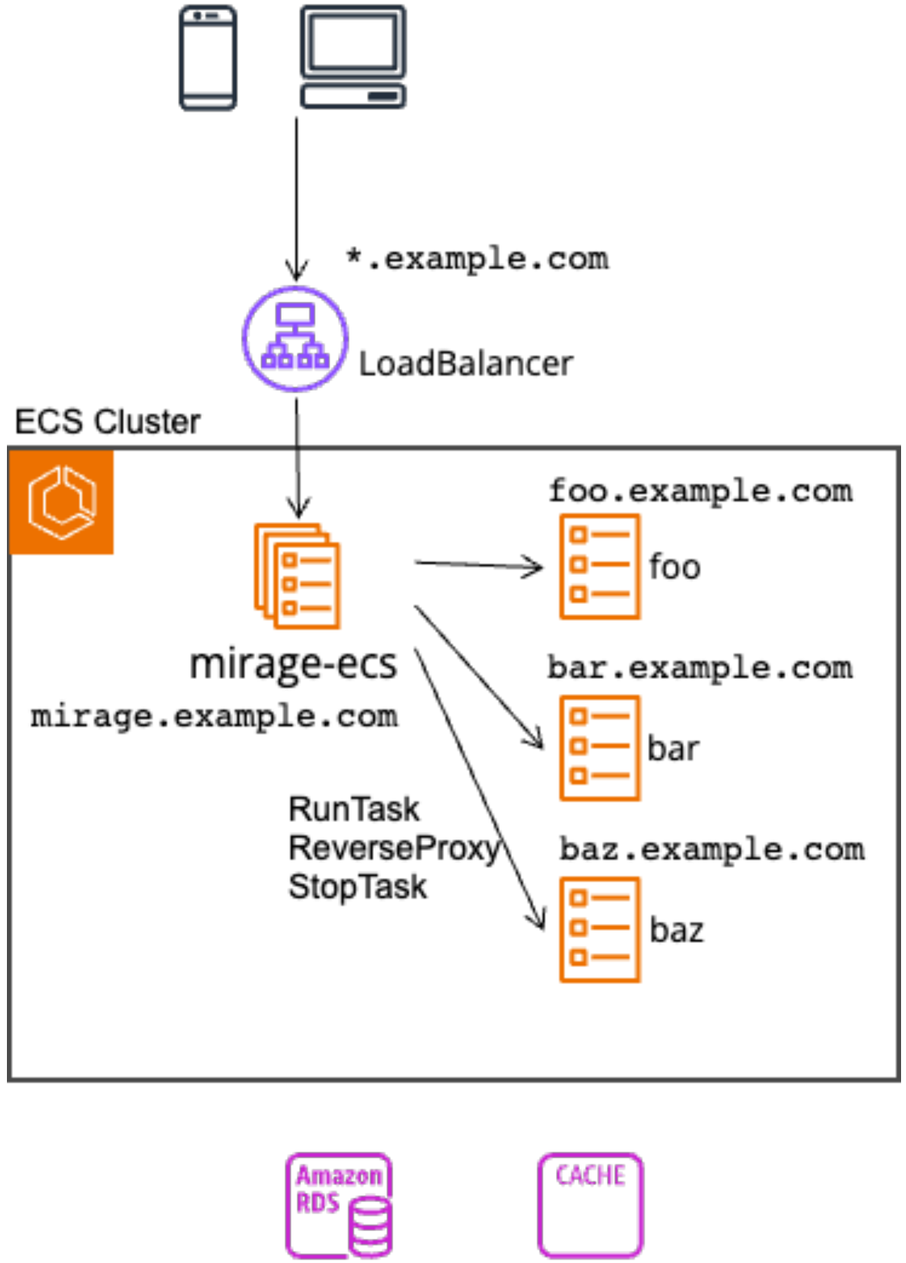
概念図

ALBに `*.example.com` を割り当てる
mirage-ecsに**全てのリクエスト**を転送
- `foo.example.com` は環境 **foo** にproxy
- `bar.example.com` は環境 **bar** にproxy

`mirage.example.com` で WebUI / API を提供
- 環境起動 → ECS `RunTask`
- 環境削除 → ECS `StopTask`

mirage-ecs とは

1.「環境」を **ECS のタスクとして起動/停止**
2.「環境」へのリクエストをタスクに
Reverse Proxyするソフトウェア



mirage-ecsを使うとなにが嬉しいか

使いたい環境を何個でも、すぐに用意できる

- ECSタスクをFargateで起動すればいくつでも
- 起動時間は最短1分〜長くても10分程度

低コスト

- 1環境 = ECSタスク1個(1〜2vCPU, Memory 4GB〜)
- ECS on EC2で動かすことも可能

環境の起動削除がWebUIとAPIで操作できる

- WebUI: 非エンジニアにも優しい
- API: 起動/削除をSlack botなどで実装できる

ChatOpsの例


fujiwara 11:02
akane launch fujiwara master
akane APP 11:02
@fujiwara fujiwara での環境立ち上げを開始します (dev) [ip-10-206-242-76.ap-northeast-1.compute.internal]
@fujiwara fujiwara に master で環境作成依頼をだしたよ！
akane APP 11:14
@fujiwara fujiwara の環境が起動したよ！
デバッグページは https://fujiwara.dev /debug で
管理画面は https://fujiwara.dev /admin だよ!
fujiwara 11:16
akane terminate fujiwara
akane APP 11:16
@fujiwara fujiwara のお掃除をお願いしたよ！！ (dev) [ip-10-206-242-76.ap-northeast-1.compute.internal]
@fujiwara fujiwara で利用しているDB,Redis,S3が削除されます (dev) [ip-10-206-242-76.ap-northeast-1.compute.i
akane APP 11:16
@fujiwara fujiwara の停止が完了したよ！

Webアプリケーションをmirage-ecsで起動する

必要なものとリポジトリが入ったイメージとECSタスク定義を用意

- Gitリポジトリをイメージに焼き込んでおく(大きなリポジトリの場合)
- 小さい場合は起動時に全部cloneしてもよい

mirage-ecsが定義する環境変数

起動時に環境名とブランチを指定すると、環境変数を設定する

- **SUBDOMAIN** : 環境名 `fujiwara`, `event-111`, ...などサブドメイン
- **GIT_BRANCH** : `main`, `feature/some-x`, ... Gitのブランチ名
- ほかにも設定で任意の環境変数を定義できる

↑ WebUI/APIから起動時に指定可能

Webアプリケーションをmirage-ecsで起動する

コンテナのentrypointで...

ブランチを切り替える

- `git (clone|fetch) && switch` で使いたいブランチに切り替える
- `git switch origin/$GIT_BRANCH` ← 起動時に指定したブランチ

DBなどの外部リソースを用意する

- 作成に時間が掛からない、従量課金なリソースはその場で作成
- `$SUBDOMAIN` を名前に含めて別のものとして作る
- DynamoDB のテーブルとかSQS queueとか

Webアプリケーションをmirage-ecsで起動する

起動に時間が掛かる、最低課金額が大きいリソースは共用する工夫

例: RDS (MySQL)

- データベースを$SUBDOMAINを元にして作る

- `DBNAME=$(echo $SUBDOMAIN | tr - _) mysql -e "CREATE DATABASE IF NOT EXISTS $DBNAME ..."`

例: ElastiCache Redis

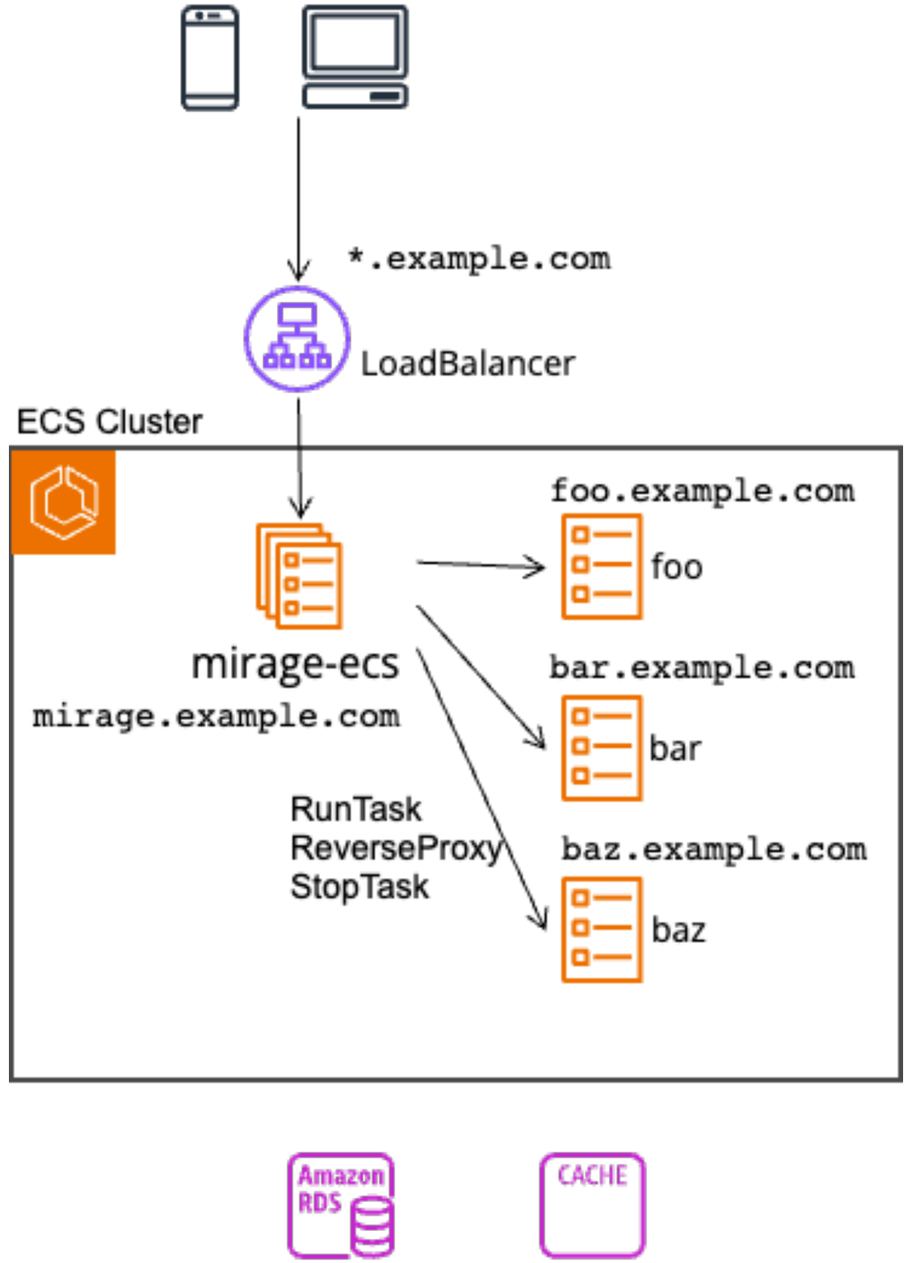
- key_prefix を `${SUBDOMAIN}:` にして論理的に分離
(クライアントライブラリの機能)

Webアプリケーションを mirage-ecsで起動

外部リソースの準備が終わったら起動

- DB migrationや初期データのimport
- ライブラリの更新 (CPAN, Gem...)、ビルド
- サーバー起動 (plackup, rails s...)

起動した環境へのReverse Proxyはmirage-ecsが自動で定義してくれます



mirage一族の歴史

2014年 mirage 誕生

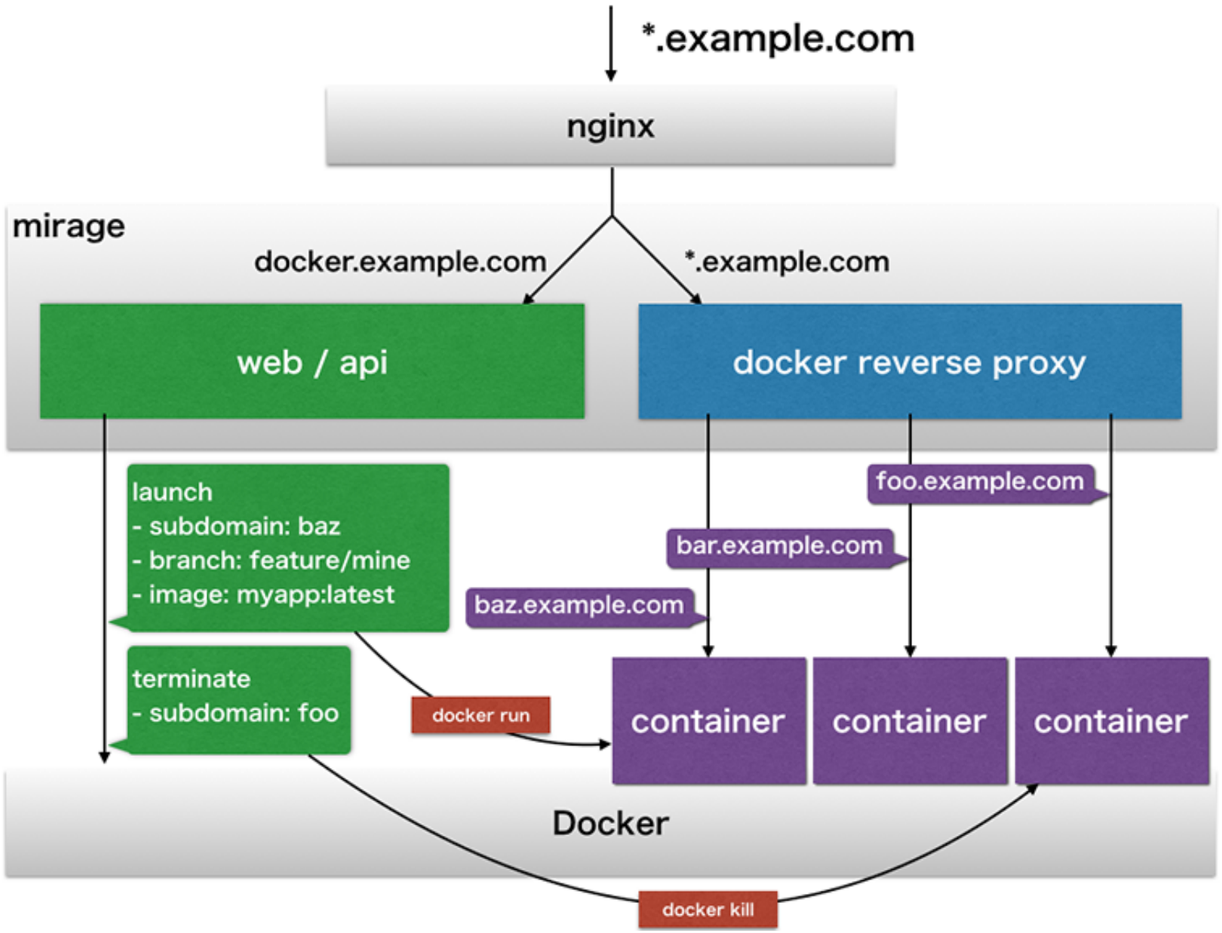
"Dockerで非エンジニアでも開発環境を上げ下げできる、
mirageというツールを作りました"

`https://techblog.kayac.com/mirage_for_docker.html`

2014年8月 **@acidlemon** が作成、リリース

ホスト上でDockerコンテナを起動して**Reverse Proxy**する仕組み

mirage 概念図


*.example.com
nginx
mirage
docker.example.com
*.example.com
web / api
docker reverse proxy
launch
- subdomain: baz
- branch: feature/mine
- image: myapp:latest
terminate
- subdomain: foo
foo.example.com
bar.example.com
baz.example.com
docker run
container
container
container
Docker
docker kill

mirageはとても便利だった

ちょっと大きめのインスタンスを用意すれば、
個別の開発環境がいくつでもすぐに起動できて最高！

しかし時は流れて2018〜新規プロジェクトで**コンテナ(ECS)環境が主流に**

2019〜 いよいよ大コンテナ時代
新規だけではなく既存のEC2のサービスも**ECSに移行していく**流れ

本番がECSになってmirageにちょっと不満が

起動する環境はホストのDockerで起動する**「コンテナ」**
mirageと別のホストでコンテナを起動することはできない
(remoteでdocker APIを叩けば論理的にはできるが
方法は用意されていなかった)

環境名/コンテナの対応を**LevelDBでローカルストレージに記録**

つまりホスト1台での動作が前提

mirageの限界 - シングルホスト構成

大量の環境を起動するには**大きな1台のインスタンスが必要**
→ CPUはある程度オーバーコミットできるがメモリは必要

起動時やマイグレーション時に**ホストのストレージを酷使する**
→ IOが速いインスタンスで動かしたい
EC2ならi3インスタンスのエフェメラルSSDとか

スケールしたいとき=動いているホストが限界に達したとき
一度環境を全部落として起動し直すしかない

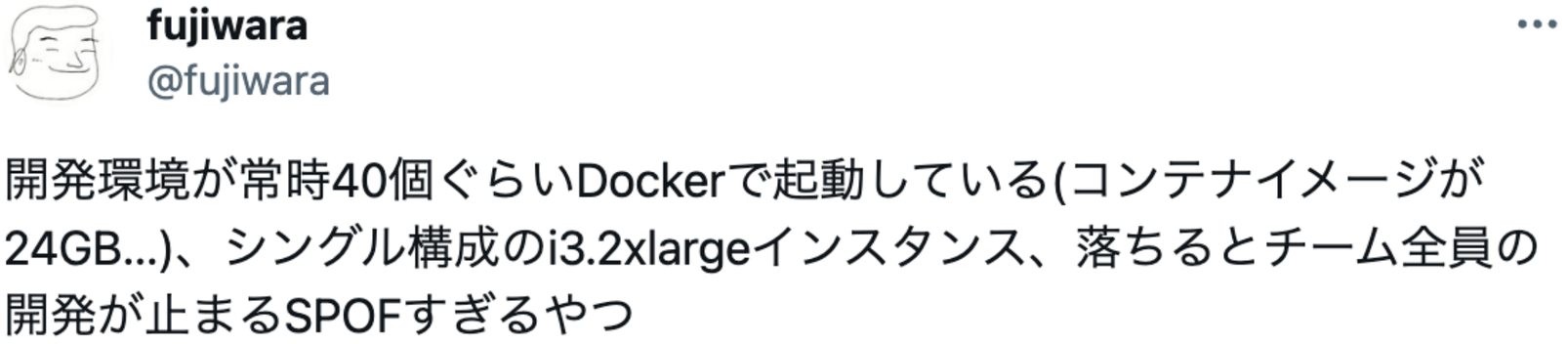
mirageの課題 - サイドカーを持てない

環境として起動する単位 = コンテナ

必要なサーバー、ミドルウェアを全て詰め込んでsupervisorで起動

foreman (Ruby) / kazeburoさんのProclet (Perl) / mattnさんのgoreman (Go)

あらゆるものが全部入りの特盛りコンテナに



2019年6月 mirage-ecs誕生

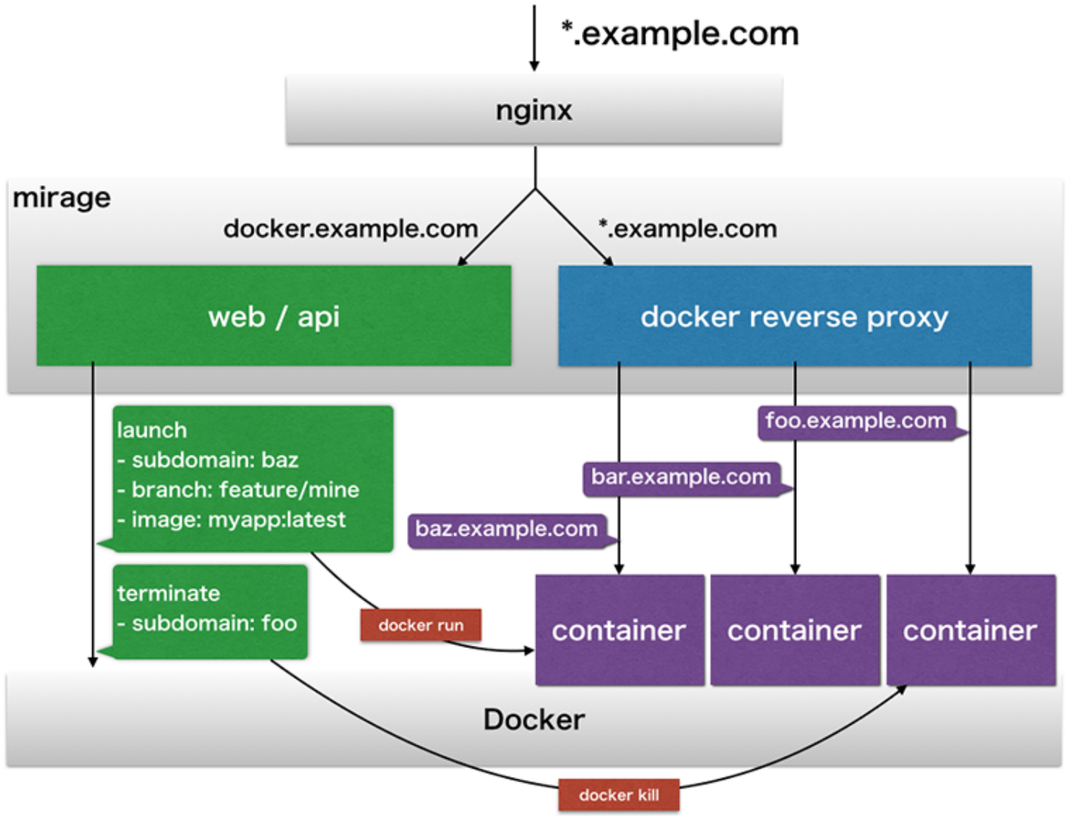
開発環境**だけ**のためにEC2を維持したくない
プロビジョニングもパッチ当ても再起動も面倒
スケーラブルなmirageがほしい!

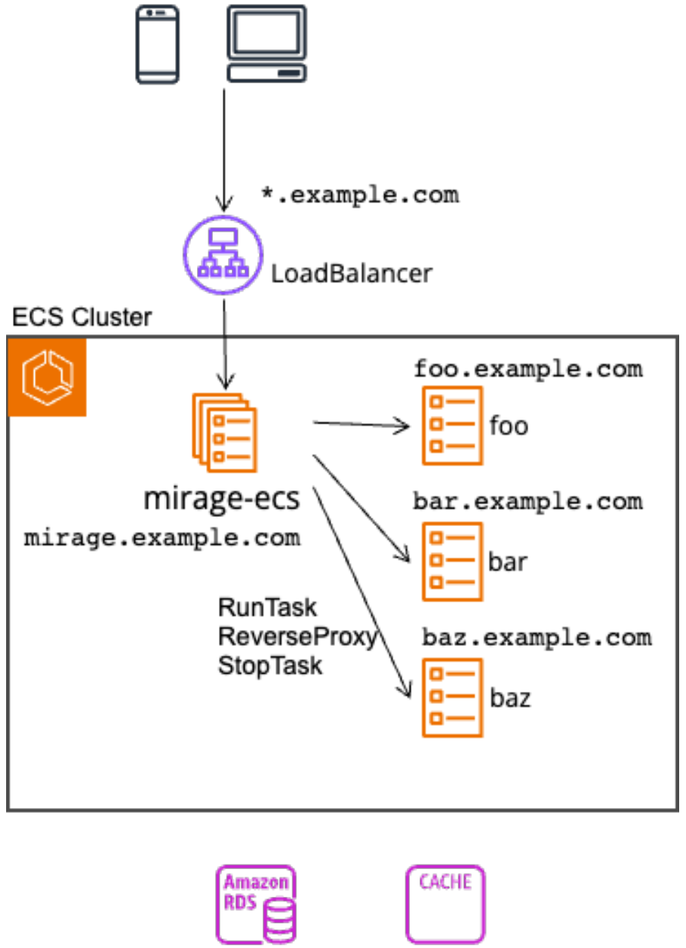
ついカッとなって

mirageをforkしてmirage-ecsを開発

余っていた有休を3日消化したらできた
その後 acidlemon/mirage-ecs にtransfer

mirage と mirage-ecs





mirage → mirage-ecs

LevelDB → **ストレージレス**

- ReverseProxyのために「環境名」->「IPアドレス」のマッピングが必要
- ECSタスクには**タグ**が持てる
- 環境名などのメタデータはタグに入れる、IPアドレスはECSが知っている
- 定期的にECS APIでタスクとIPアドレスを取得してReverseProxy定義を更新

サイドカーを持てる

- 「環境=コンテナ」→**「環境=ECSタスク(コンテナ最大10個)」**

mirage → mirage-ecs

シングルホスト構成 → **複数台構成にできる**

- 環境はmirageが動作するホスト上ではなく独立して起動
- mirage-ecs自体はストレージレスなので複数台で動作できる
  とはいえProxyしかしないので100環境あっても1台で十分
  デプロイもrollingでOK
- 環境とはVPC内で通信するので複数台で動作できる

mirage → mirage-ecs

1台のインスタンス+mirage(プロセス)+複数のDockerコンテナ(プロセス)
から

複数のインスタンス(ECSクラスタ) + mirage-ecs(ECSサービス)

+ 複数の環境(ECSタスク)

インスタンス → クラスタ / プロセス → タスク

スケーラビリティを獲得するためのコンテナ技術に乗った正統進化

もうすぐ10歳になるサービスも移行しました

7年間運用したソーシャルゲームを
Amazon EC2構成からAmazon ECS構成へと
乗り換えた話

https://techblog.kayac.com/2021/12/20/120000
https://commojun.github.io/yapc2022/

mirage-ecsの設計と実装

mirage-ecs = Go製のWeb APIサーバー兼reverse proxy

メインのHTTPサーバー: リクエストを全部受けてHostヘッダをみる

- `mirage.*` → アプリケーションハンドラを実行
- それ以外 → Hostに対応した環境宛てのReverseProxyを実行

その他のgoroutine worker

- ECS APIを叩いてReverseProxyを更新するworker
  「環境=タスク」が増えたらReveseProxyを作成してmapに追加
  減ったらReveseProxyをmapから削除
- リクエスト数集計をCloudWatchに投げるworker

環境ごとのreverse proxyの実装

github.com/methane/rproxy
WebSocket対応の単一ホスト宛のReverseProxy実装ライブラリ

11年前に作られたもの(その後更新はないが…)

Goの標準モジュールのみでできているので

Go 1.22(2024年)でもちゃんと動く！！

ALBもWebSocketに対応しているので

何も考えずにWebSocketが動きます

ストレージレス

管理するものを減らしてデプロイを容易にするため

mirage-ecsはECSサービスをデプロイすれば動作する

- 初代mirageはストレージにLevelDB(KVS)を使っていた
- AWS専用なのでDynamoDBを使うことも考えたが……
- ECSのタグに情報を持つことでストレージレスに
 環境はせいぜい100+個程度なのでこれで十分

人は永続ストレージを管理したくない

コンフィグレス

初代mirageで必要だった設定はこれだけ

```
host:
  webapi: mirage.example.com
  reverse_proxy_suffix: ".example.com"
listen:
  http:
    - listen: 80
      target: 80
```

これぐらいなら手で書いても問題ないが...

ECSではタスクを起動するための設定が(どうしても)必要

```
ecs:
  region: "ap-northeast-1"
  cluster: mycluster
  default_task_definition: myapp
  enable_execute_command: true
  launch_type: FARGATE
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-aaaa0000
        - subnet-bbbb1111
        - subnet-cccc2222
      security_groups:
        - sg-11112222
        - sg-aaaagggg
      assign_public_ip: ENABLED
```

自動設定機能を作った

mirage-ecs自体が**ECSで動く前提**なので**自分自身の設定を流用する**
- タスクメタデータAPIで自分が動いているクラスタ名とタスクIDを取得
- タスクIDからECSサービス名を取得 (なぜかこれはメタデータにはない)
- ECSサービスのNetworkConfigurationなどを取得

設定ファイルのECS設定が特にない場合、デフォルトでこれを使う

人は設定ファイルを書きたくない

mirage-ecsの実践的な運用

mirage link

ECSタスクには**10コンテナまで**含められるが**足りなくなって困った**ので...

1. 起動時に複数のタスク定義(A,B)を渡す(ユーザー)
2. それぞれのタスク定義からタスクa,bを起動する (mirage-ecs)
3. コンテナ名とタスクのIPアドレスでRoute53に名前を定義(mirage-ecs)

`nginx.foo.example.com`	タスクaのIPアドレス
`webapp.foo.example.com`	タスクaのIPアドレス
`backend.foo.example.com`	タスクbのIPアドレス
`sidecar.foo.example.com`	タスクbのIPアドレス

複数のタスクを1つの「環境」として扱える

アクセス数計測と定期削除

比較的低コストとはいえ、環境数に比例してコストは掛かる

使っていない環境を自動的に停止したい！

- `http.Transport`をカスタマイズしてReverseProxyに
 アクセスカウンターを実装
- 「環境」単位で1分あたりのHTTPアクセス数を記録
- 毎分CloudWatchにMetricsとして送信

これで**「一定期間アクセスがない(=使われていない)環境」**が分かる

POST /api/purge

```
{
  "excludes": ["foo", "bar"],
  "exclude_tags": ["branch:preview"],
  "duration": 86400
}
```

excludes: 特定の環境名は除外

exclude_tags: 指定したタグが付いている環境は除外

duration: 過去指定した秒数にアクセスがない環境を削除する

止めたくない環境もあるので除外条件を指定可能(顧客向けとか)

環境が起動できない理由を知りたい

ECSタスク、立たないとき困りますよね...

github.com/fujiwara/tracer
"Amazon ECS タスクのイベントとログを時系列で出す tracer を作った"[4]

- タスクに関連するイベント(作成、起動開始、pull開始と停止、停止開始、停止完了など)
- タスク内のコンテナが CloudWatch Logs に出力したログ
- (ECS サービスから起動されたタスクの場合) サービスのイベントログ

これをmirage-ecsのWebUIからすぐ見られるように

---

[4] https://techblog.kayac.com/ecs-task-tracer

tracerの出力例 (起動→停止)

```
2024-01-31T07:05:17.529Z    TASK    Created
2024-01-31T07:05:32.718Z    CONTAINER:nginx LastStatus:PENDING HealthStatus:UNKNOWN
2024-01-31T07:05:32.718Z    TASK    LastStatus:PENDING
2024-01-31T07:05:22.775Z    TASK    Connected
2024-01-31T07:05:32.391Z    TASK    Pull started
2024-01-31T07:05:39.561Z    TASK    Pull stopped
2024-01-31T07:05:39.590Z    TASK    Started
2024-01-31T07:05:40.070Z    CONTAINER:nginx LastStatus:PENDING HealthStatus:UNKNOWN
2024-01-31T07:05:40.070Z    TASK    LastStatus:PENDING
2024-01-31T07:05:40.070Z    CONTAINER:nginx LastStatus:RUNNING HealthStatus:UNKNOWN
2024-01-31T07:05:40.070Z    TASK    LastStatus:RUNNING
2024-01-31T07:05:39.573Z    CONTAINER:nginx /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
2024-01-31T07:05:39.573Z    CONTAINER:nginx /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
2024-01-31T07:05:39.576Z    CONTAINER:nginx /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

-- (略) --

2024-02-04T01:03:04.797Z    TASK    LastStatus:STOPPED
2024-02-04T01:02:50.022Z    TASK    Stopping
2024-02-04T01:02:50.022Z    TASK    StoppedReason:Terminate requested by Mirage
2024-02-04T01:02:50.022Z    TASK    StoppedCode:UserInitiated
2024-02-04T01:02:52.015Z    CONTAINER:nginx 2024/02/04 01:02:52 [notice] 1#1: signal 15 (SIGTERM) received, exiting
2024-02-04T01:02:52.018Z    CONTAINER:nginx 2024/02/04 01:02:52 [notice] 34#34: exiting
2024-02-04T01:02:52.018Z    CONTAINER:nginx 2024/02/04 01:02:52 [notice] 35#35: exiting
2024-02-04T01:02:52.018Z    CONTAINER:nginx 2024/02/04 01:02:52 [notice] 34#34: exit
```

tracerの出力例 (起動失敗)

```
2024-02-04T01:09:02.676Z    TASK    Created
2024-02-04T01:09:08.180Z    TASK    LastStatus:STOPPED
2024-02-04T01:09:08.180Z    TASK    LastStatus:DEPROVISIONING
2024-02-04T01:09:08.180Z    TASK    LastStatus:PROVISIONING
2024-02-04T01:09:06.503Z    TASK    Connected
2024-02-04T01:09:21.109Z    TASK    Execution stopped
2024-02-04T01:09:31.147Z    TASK    Stopping
2024-02-04T01:09:31.147Z    TASK    StoppedReason:CannotPullContainerError:
 pull image manifest has been retried 1 time(s): failed to resolve ref docker.io/library/nginx:lates:
 docker.io/library/nginx:lates: not found
2024-02-04T01:09:31.147Z    TASK    StoppedCode:TaskFailedToStart
```

imageをpullできていないため

StoppedCode:TaskFailedToStart なのがすぐ分かる

他にも便利機能がいろいろ

認証認可とか、環境名のワイルドカード対応とか...

詳しくは **github.com/acidlemon/mirage-ecs** をどうぞ

まとめ

「ブランチ別開発・検証環境」があると開発効率が上がります
が、実装方法によってはコストが大きくなりがちです

acidlemon/mirage-ecs は独立した「環境」をECSタスクとして起動する
少ないコストで、いくつでも高速に検証環境を起動できる OSS です