
aws
SUMMIT
ONLINE
JAPAN

SessionID

インフラエンジニアは動かない

- モンスターハンターライダーズ 編-

中島　淳平
システムリード
株式会社カプコン

中村　一樹
データリード
株式会社カプコン

aws SUMMIT ONLINE

用語統一(※ガイドラインに則るため)

AWS：アマゾンウェブサービス(AWS)

ALB：Application Load Balancer

Aurora：Amazon Aurora

ASG：AWS Auto Scaling Group

DynamoDB：Amazon DynamoDB

EC2：Amazon Elastic Compute Cloud

ECS：Amazon Elastic Container Service

ECR：Amazon Elastic Container Registry

EKS：Amazon EKS

ElastiCache：Amazon ElastiCache

ELB：Elastic Load Balancing

IAM：AWS Identity and Access Management

KDS： Amazon Kinesis Data Streams

KDF： Amazon Kinesis Data Firehose

Lambda：AWS Lambda

RDS：Amazon Relational Database Service

Route 53：Amazon Route 53

S3：Amazon Simple Storage Service

SQS： Amazon Simple Queue Service

VPC：Amazon Virtual Private Cloud

モンスターハンターライダーズ

Monster Hunter Riders ni tuite

aws SUMMIT ONLINE

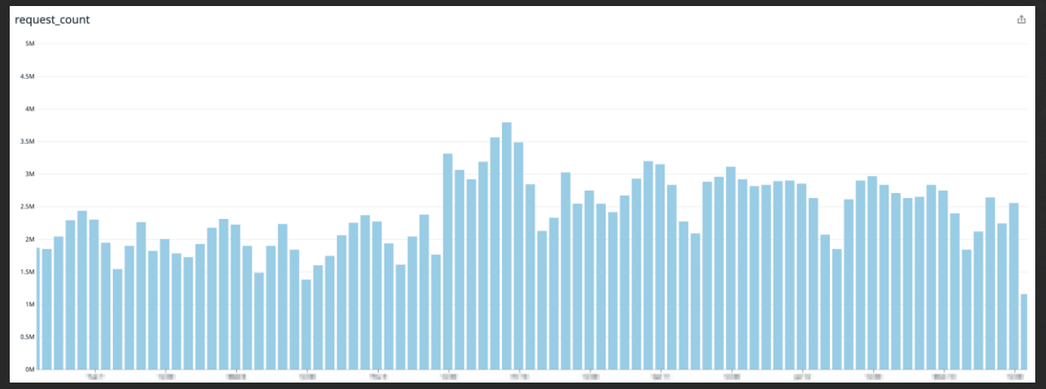
モンスターハンターライダーズ

ジャンル：RPG

プラットフォーム：iOS,Android

モンスターハンターライダーズ

DL数 500万↑



DAU 30万↑

今回はなすこと

Nani wo hanasu no.

aws SUMMIT ONLINE

Profile

今回話すひと

中島 淳平

システムリード

- 2013年頃からサーバーエンジニアとなる
  （それ以前はクライアントを担当していた）
- 「モンスターハンターエクスプロア」から
  システムリードとして
  ゲーム全体のインフラ構成に関わる
- 「モンスターハンターライダーズ」にて
  企画立ち上げからテックリードとして参加
- 障害を起こしたり
  緊急メンテしたりしながら現在に至る

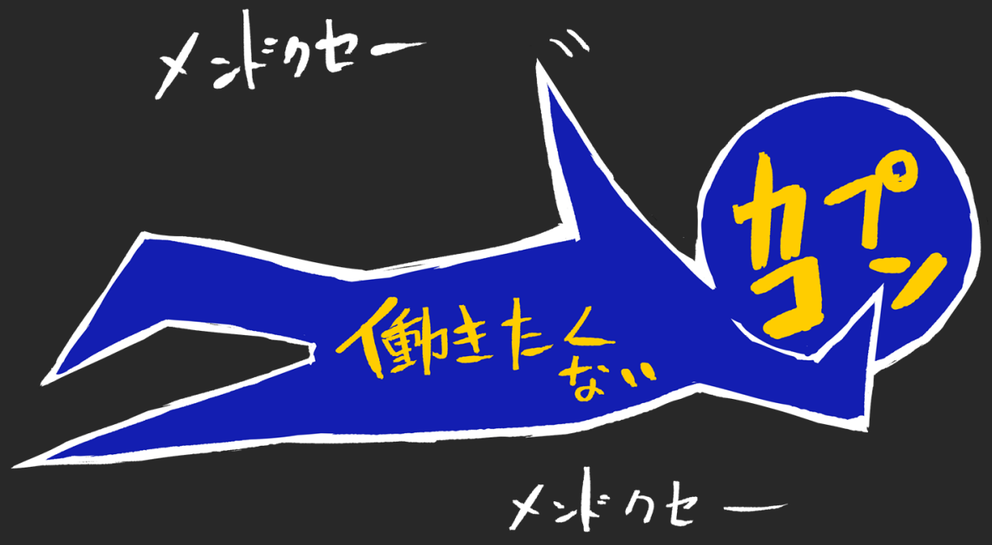
これからのインフラの話をしよう

Hatarakitaku nai de gozaru.

aws SUMMIT ONLINE

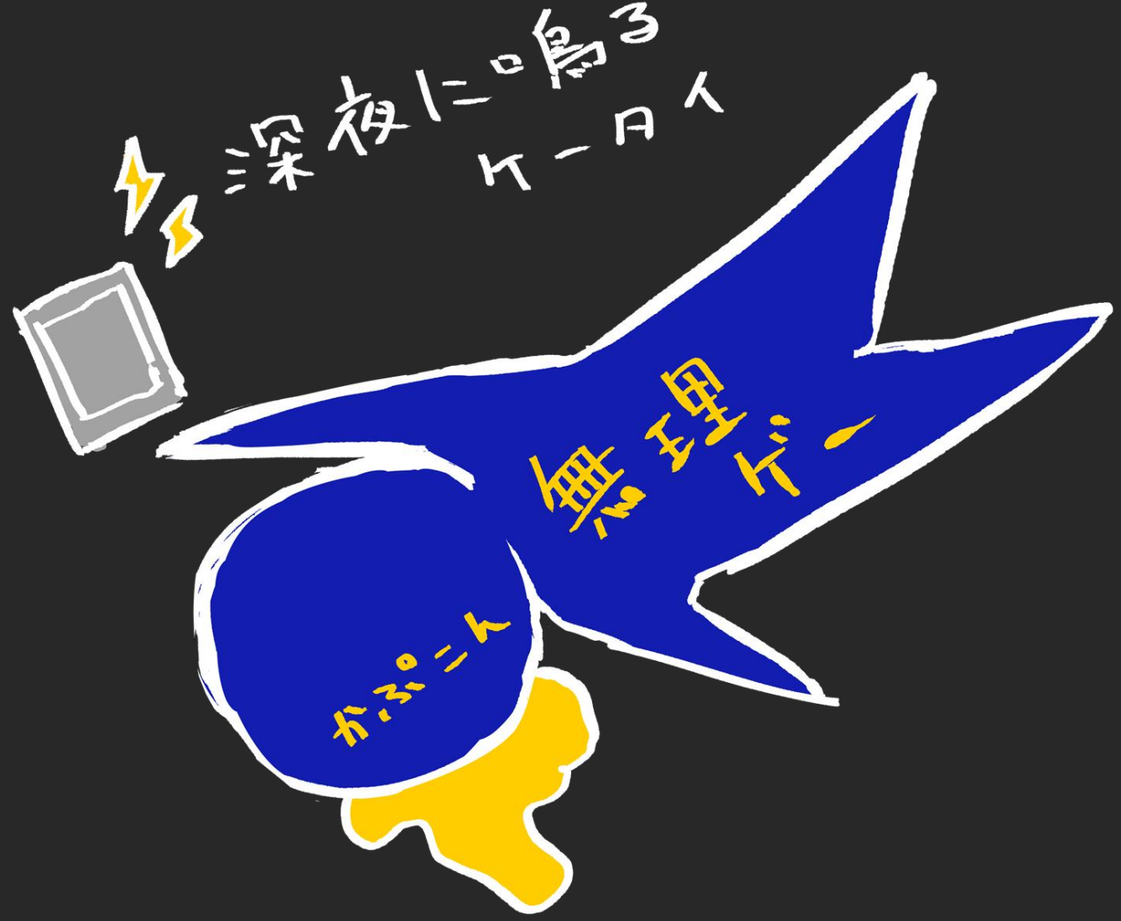
働きたくない、とても働きたくない

- （建前）管理の人件費を下げてトータルのコストを下げたい！
  - ぶっちゃけインフラの構築めんどくさい
  - 管理もめんどくさいし運用はもっとめんどくさい
  - ボタン押すのすらめんどくさいし、グラフを見るのもめんどくさい



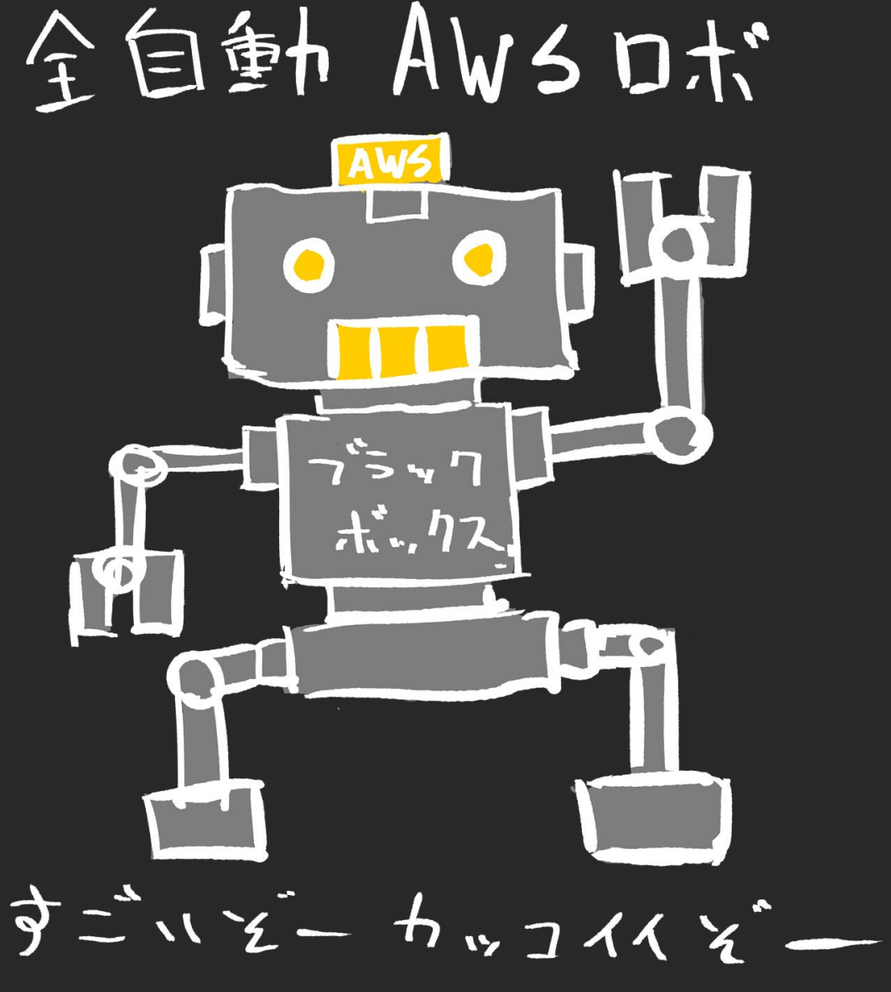
今までのインフラ

- 値段は安いほうが良い
- 性能は高いほうが良い
- すべてを把握して管理したい



全力で「働かない」インフラ

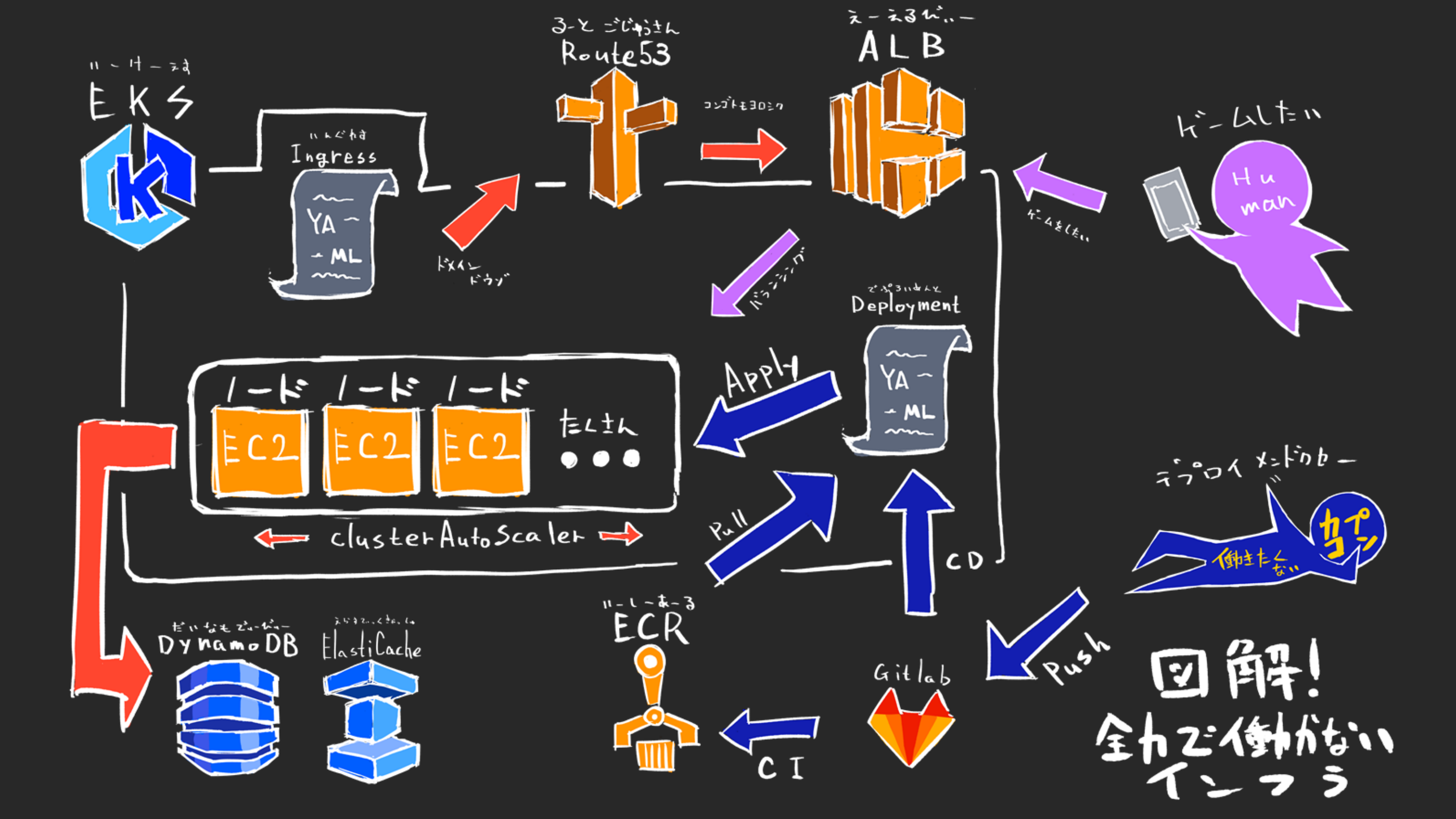
- 単価よりも管理コスト
- 性能よりも管理コスト
- ブラックボックス上等
  - インフラは不思議な力で動いている
  - ベンダーロックイン？知らんな



どういう構成になったの

kousei oshiete.

aws SUMMIT ONLINE


いーけーえす
EKS
Ingress
YA
ML
ドメイン
ドウゾ
るーと ごびあさん
Route53
コンゴトモヨロシク
えーえるびー
ALB
ゲームしたい
Human
ゲームをしたい
トランシーバ
でぷろいめんと
Deployment
YA
ML
Apply
ノード ノード ノード
EC2 EC2 EC2
たくさん
clusterAutoScaler
Pull
CD
デプロイ メンドウセー
カプン
働きたくない
だいなもでぃーびー
DynamoDB
えらすてぃくきゃっしゅ
ElastiCache
いーしーあーる
ECR
Gitlab
CI
Push
図解!
全力で働かない
インフラ

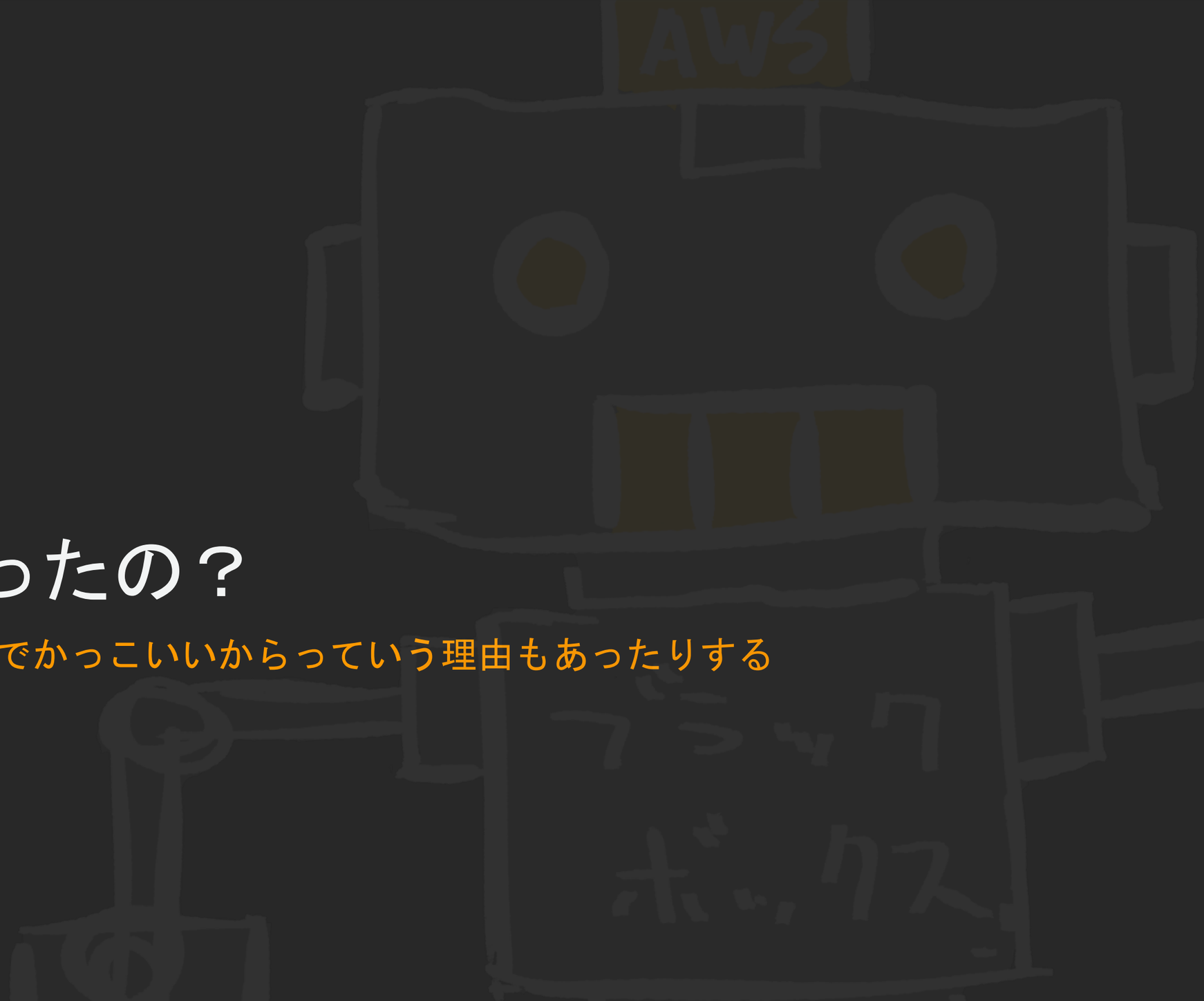
なぜEKSなのか

EKS nande.

aws SUMMIT ONLINE

ECSではダメだったの？

EKSの方がなんとなく新しそうでかっこいいからっていう理由もあったりする



そもそも守備範囲が違う

- ECSはコンテナの実行環境“しか”提供しない
  - バランシング、デプロイについては他のサービスやツールを併用しなければならない
  - 作り込みたい時はメリットだが私はめんどくさがりである
- EKSはインフラ全体を包括的に扱う
  - 各オペレーションはあくまでKubernetesのコマンドに則った形で行う

ECS

AWSから、ECSを操作する

EKS

EKSから、AWSを操作する

つまり、EKSの世話だけをしておけば幸せになる

具体的な事例

何がどう楽になったのか

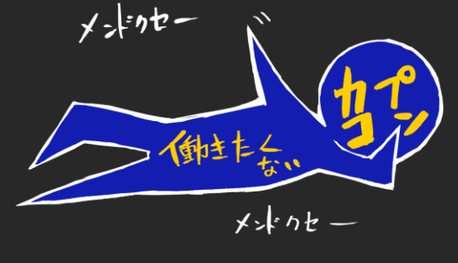
デプロイ

- ELBからEC2を切り離す
- EC2に対してコードを配布
- 依存ライブラリのインストール
- キャッシュ構築
- ELBにEC2を戻す
- 以上をすべてのEC2に対して実行

- `kubectl apply -f xxx.yml`

いままで

げんざい



働かない！しあわせ！！

Job実行

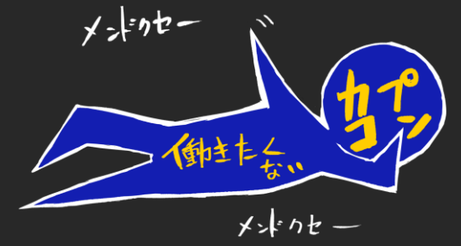
- Rundeckを構築
- Jobのコード記述
- 実行

- `kubectl apply -f xxx.yml`

いままで

げんざい

働かない！しあわせ！！



スケーリング

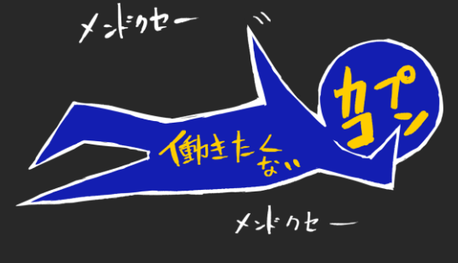
- ASGを構築
- AMIを作成
- トリガーを検討
- しきい値の調整

- `kubectl apply -f xxx.yml`

いままで

げんざい

働かない！しあわせ！！



真面目なまとめ

- もともとECSで展開する予定だった
  - EKSが日本でサービスインしたので切り替えた
    - コンテナイメージ自体は同じなので、切り替えはすぐ出来た
- EKSの後ろには結局AWSがいるのでAWS力は必要
  - 主にトラブルシュート。逆に言えば今までのノウハウは無駄にならない
  - ただ、敷居はかなり下がる。AWSコワクナイ。
- EKSはECSと用途が違う。上位互換ではない。
  - 併用するというのも全然ありだと思います。
- 割り切りも必要
  - すべてを把握したい気持ちはわかる。だが、それは本当に必要なことだろうか。
    - なんかよくわからんけどいい感じに動いている、という現実を受け入れれば楽になる。
      - （実際は使ってるうちにわかってくるけどね...）

DBだって楽したい

DB kanritte mendkusai yone.

aws SUMMIT ONLINE

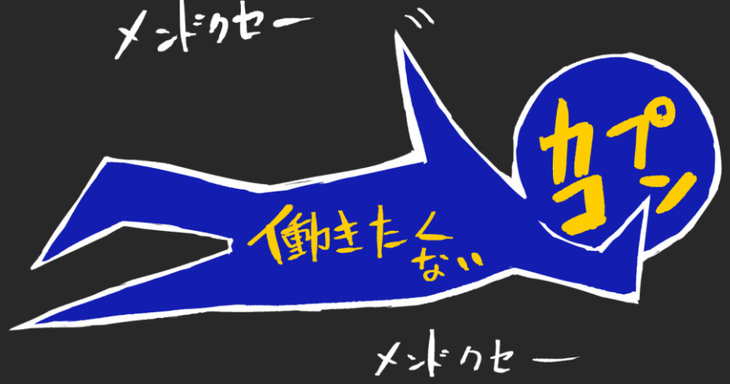
インフラ管理で一番めんどくさいこと

あえて言おう、DBの管理であると。

DBの管理は何故めんどくさいのか

- 基本的に落とせない
- サービスを続ける限りストレージが増大していく
- サービスを続ける限り性能も劣化していく
- スケールアウトめんどくさい、構成によっては出来ない
- スケールアップもめんどくさい、構成によっては出来ない
- その癖、虚弱体質ですぐ死ぬ

つまりめんどくさい



Auroraではダメだったの？

- 【前提】DynamoDBとAuroraは前提として用途、目的が違う
- とりあえずMySQL（互換含む）にしておけばよいという風潮
  - 気持ちはわかる
- 本当にMySQL（RDB）でなければならないのか？
  - RDB固有の機能は使っている？本当に？
- 慣れてるメンバー多いから...という理由だけで選んでいないか？
  - 「私SQL書けます！」という言葉は信じてはいけない
    - とんでもないクエリを書く人は（残念ながら）存在する

実はRDBじゃなくてもよいのでは？

DynamoDB

Hatsudenki.

aws SUMMIT ONLINE

「働かない」観点から見たDynamoDB

で、私はどれだけ楽ができるんだい。

- 結局、ゲームにおけるDBはセーブデータの延長であることが殆ど。
- どれくらい使うのか？をコンソールで入力するだけ
  - スペック課金ではないところがミソ。
    - とりあえず適当に設定しといて数値見つつ調整すれば良い
- VPCがー、とかサブネットがーとか気にしなくて良い
  - ぶっちゃけネットワークとかめんどくせー（暴論）
- ユーザーがーとかアクセス権限がーとか気にしなくて良い
  - IAMで許可されていればOK、そうでなければNG。シンプル。
- スキーマをあまり気にしなくて良い
  - スキーマレスなので、主キー以外をあまり気にする必要がない

DynamoDBを管理するということ

≒CapacityUnitをいい感じに調整すること

本当にこれだけ。いや本当に。

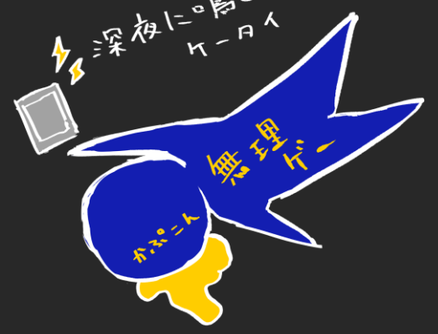
本当に楽だったのか

そんなうまい話はない

スキーマ設計がむずい

適当にやるとしぬ

- よくやる例
  - SQL感覚でテーブルを正規化してしまう
    - 1テーブルに対してしかクエリが発行できないのでクエリ発行数が増えて死ぬ
  - 1レコードに何でもかんでも突っ込んでしまう
    - レスポンス容量がでかくなって死ぬ（1クエリで返却できる量に制限がある）
      - 1レコードの容量制限もあるが、これは滅多に超えることはないです
  - スキーマレスだからという理由でスキーマを決めずに実装する
    - 口伝のスキーマ伝説が生まれる
      - 属人性っていうレベルじゃねえぞ
    - ただ、厳密に決めすぎてもメリットが薄まる



Solution.

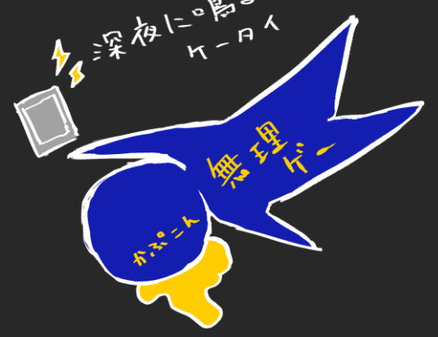
もんだいかいけつ

- オレオレDSLを定義して運用した
  - Kubernetesのマニフェストに合わせてYAML形式で記述するようにしてある
  - 流し込むといい感じに解釈してテーブルを作ってくれる
    - ついでにPythonのModelクラスコードも生成する

クエリが念仏レベルでわからん

南無

- SQLとあまりに文化が違いすぎる
- 辞書配列に突っ込む形なので型ヒントの恩恵をほぼ受けられない
- IDEによる補完もない
- というかboto3使いづらい
  - IDE補完くらい効かせてくれないですかね
- OSSライブラリによる支援もあまり期待できない
  - 無くはないが洗練されているとは言えない



Solution.

もんだいかいけつ

- オレオレORMを作った
  - OSSのORMも存在はするが、残念ながら目的に合わなかった
  - DynamoDBはKVS的な使い方しか出来ないのでラップするのはそれほど難しくない
  - 前述のオレオレDSLと連動してモデルクラスを自動生成する
- 難しいことは基本的にORMで全部隠す
  - 書けるけど書いてほしくないこと、は結構あります

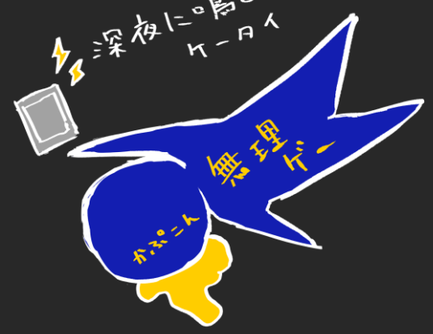
興味ある方はGithubで公開してるのでぜひ（宣伝）

https://github.com/Jumpei-Nakajima/hatsudenki

CapacityUnitの管理が手間

テーブルごとに決める必要がある

- テーブルをガンガン増やすとCUの管理が複雑になって死ぬ
  - テーブルの個数自体に課金は無いのでついつい増やしがち
  - AutoScaleするにしても、MIN値とMAX値を適切に設定しないと無駄な課金が発生する
- AutoScaleはそれほど反応が早くない
  - スパイクで簡単に死ぬ
- オンデマンドもそれほど反応が早くない
  - オンデマンドでもCUスロットルは起こります
  - AutoScaleよりはマシ、というレベル。
    - スケールに備えるというより、無駄な課金を発生させないための仕組みと考えるべき



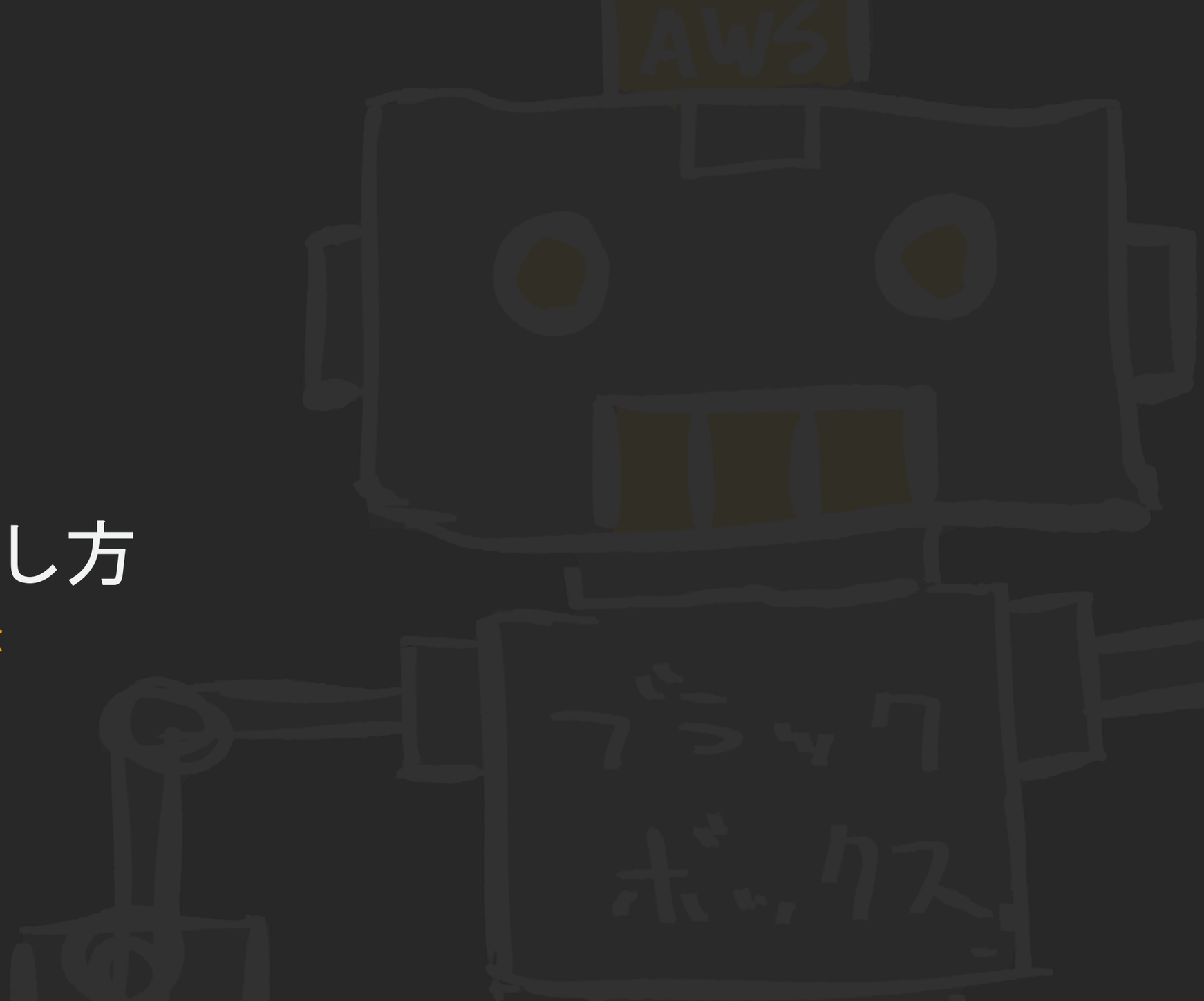
Solution.

もんだいかいけつ

- テーブルの数を絞る
  - 絞ると言ってもどうやって？

テーブルの減らし方

テーブルをまとめるという概念



隣接関係リスト設計

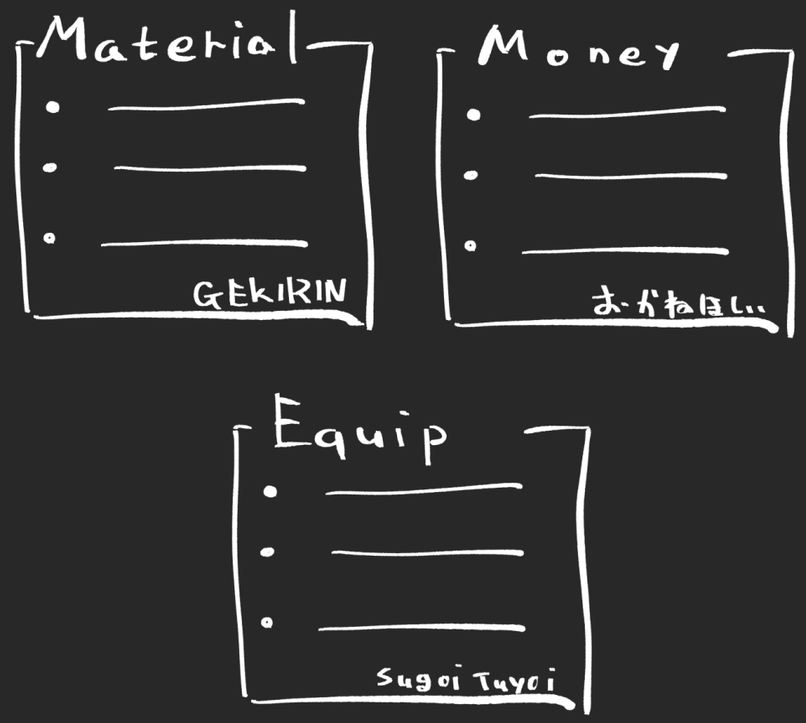
関係のあるデータを近くに置くという考え方

- 用途が似通っているものは一つのテーブルにまとめましょう
  - 例）ユーザーの所持物はすべてBoxテーブルにまとめる
- 用途が似通っているものは主キーも大体同じになる
  - ゲームにおける主キーはほぼすべてユーザーID(HASH)、固有ID(RANGE)である
    - つまりゲームと隣接関係リスト設計は非常に相性が良い。
      - その気になれば多分1テーブルでサービス提供も可能（流石にやらないけど）
- DynamoDBは主キー以外は定義しない
  - スキーマレスだからね

具体的に

For example.

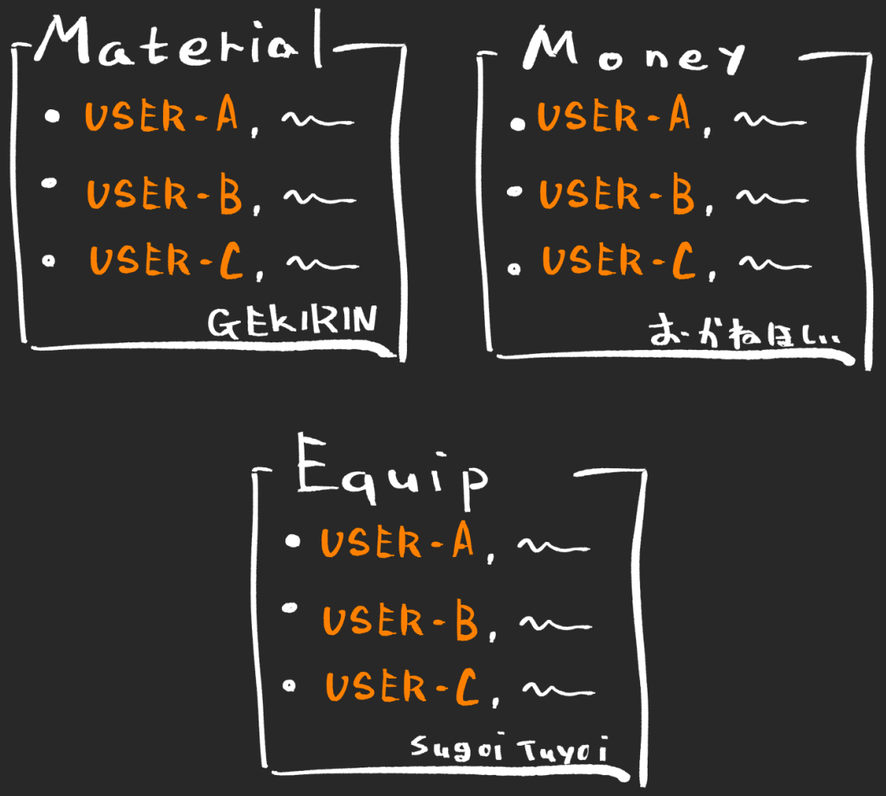
- ユーザーの所持物（Box）を考える
  - ユーザーの所持物は種類がたくさん。例として通貨(money)と装備(equip)と素材(material)



Hashキーはどれもユーザーid

ユーザーの所持物だからね

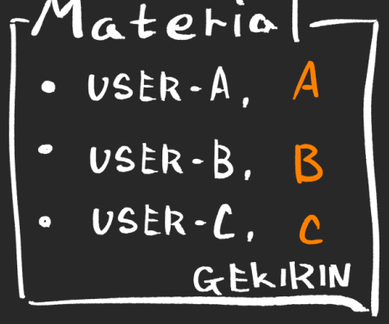
- ゲームにおいて、大抵のテーブルの主キーはユーザーIDになる
  - カーディナリティ的にも一番都合が良い

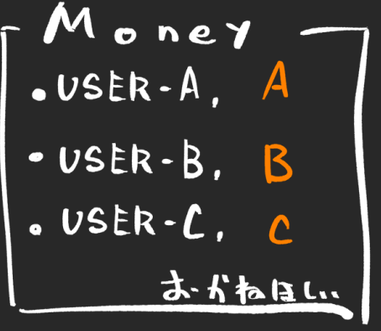


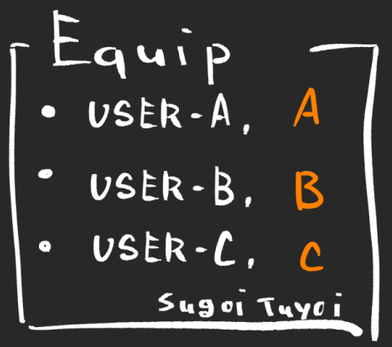
Rangeキーはテーブルによって違う

違うけど意味は同じ

- 通貨であれば通貨ID
(有償or無償,android or iOS,等)
- 装備であれば装備ID
(飛竜刀[紅葉],飛龍刀[朱],飛龍刀[楓],等)
- 素材であれば素材ID
(火竜の紅玉,火竜の逆鱗,等)
- カテゴリ毎にユニークなIDという点
では共通している



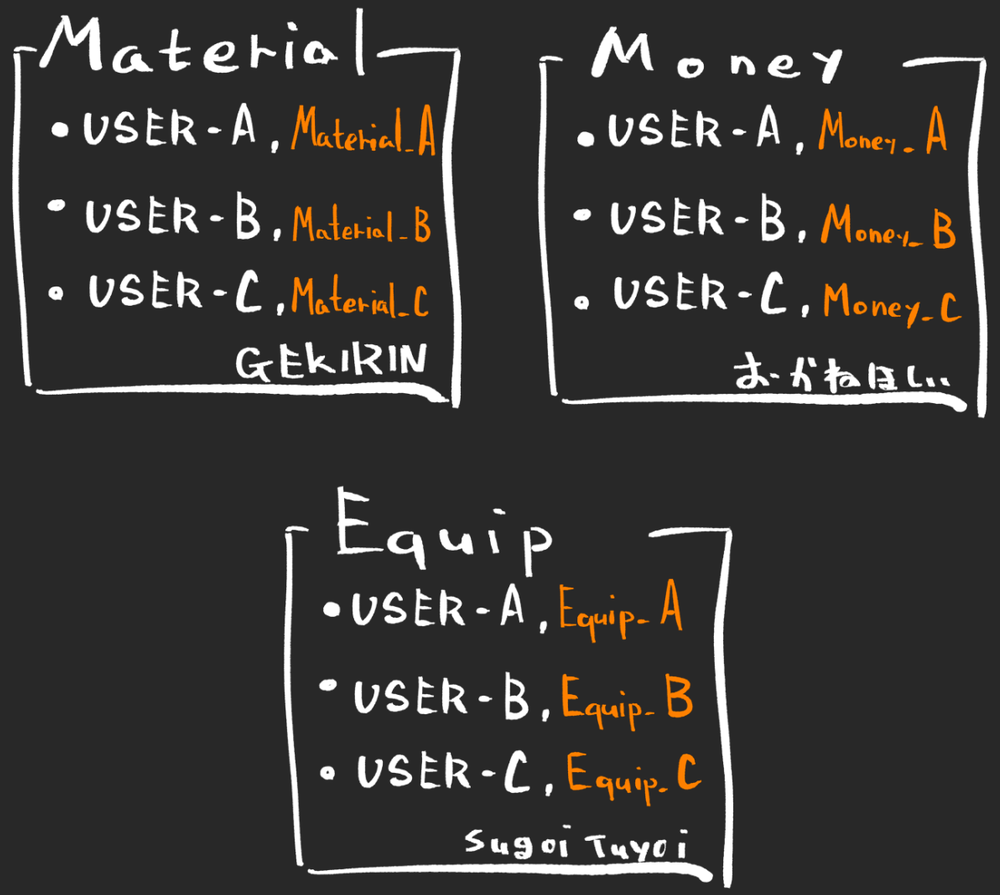




一つのcolumnで複数の種類を表現する

プリフィックス

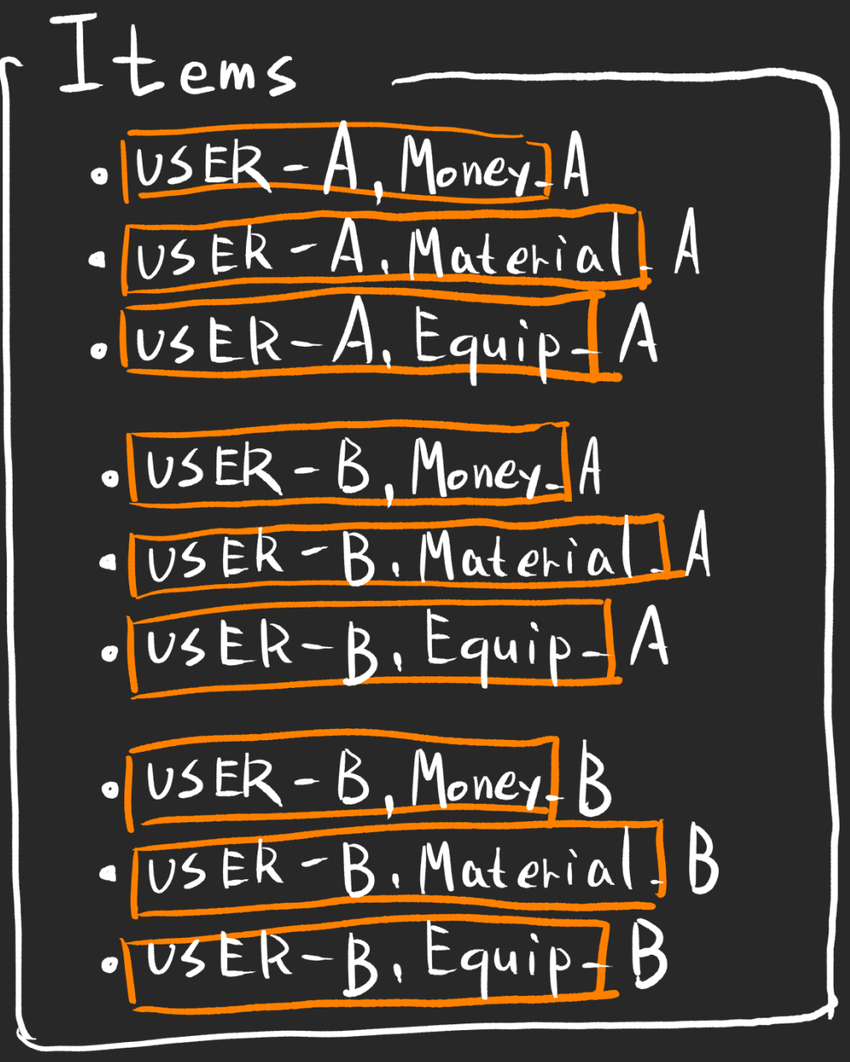
- カテゴリ毎にユニークIDは即ち{カテゴリ}_{ID}で表現可能
  - 例）material_火竜の逆鱗
  - ※実際には'_'はID部にも使われることが多いため別の記号の方が良い。'@','/'とか。
    - ちなみに日本語も問題なく使える



クエリで検索する時はbegins_with構文を使う

Rangeキーはソートされているのがミソ

- Rangeキーはその内容でソートされている
ことが保証される
  - つまり、前方一致検索やbetween検索が可能
- Rangeキーには
{カテゴリ}_{ID}で定義されているので
  - ユーザAの素材をすべて取得したい
→hash="user_A", range=begins_with("material_")
  - 特定ID(火竜の逆鱗)を取得したい
→=hash="user_A", range="material_火竜の逆鱗"

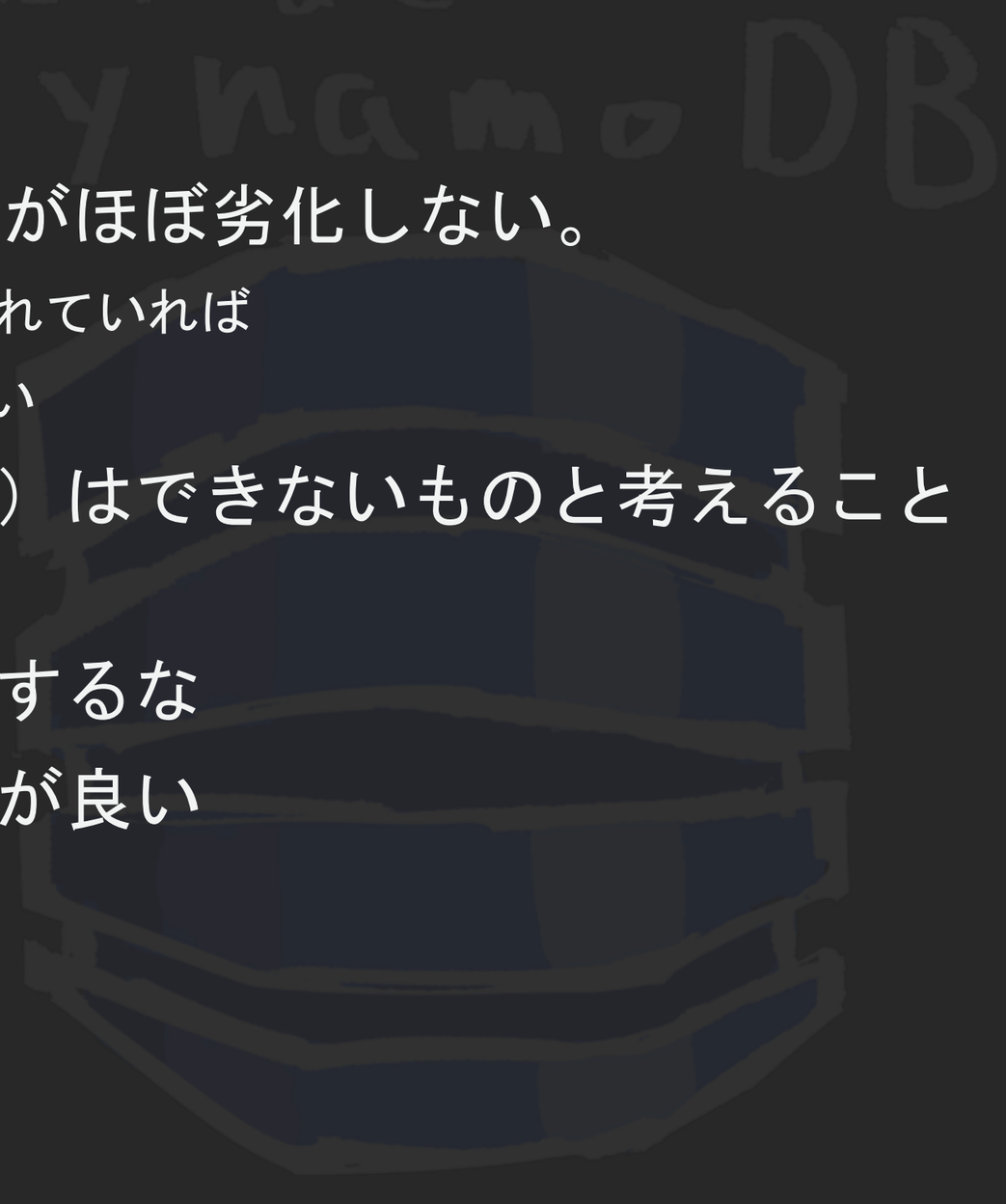


Well done!!

性能劣化について

ない

- DynamoDBはレコード数が増えても性能がほぼ劣化しない。
  - ハッシュキーのカーディナリティがきちんと担保されていれば
    - ゲームにおいてはユーザーIDにしておけば問題ない
- ただし、キーを利用しない検索（全検索）はできないものと考えること
  - 100万レコードから1レコード探り当てるとか死ぬ
- 容量もほぼ無限にスケールするから気にするな
- むしろガンガンレコード入れていった方が良い



真面目なまとめ

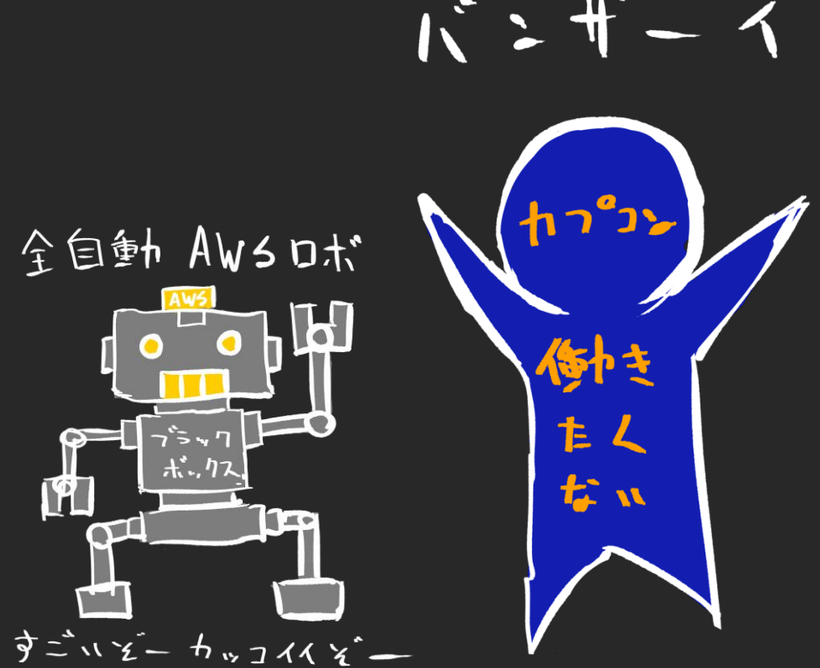
どのような結果

- MySQLを止めたことで失ったものは多かったが得たものも多かった
  - 一度立ち止まって「本当にRDBMSは必要か」を考えてみて欲しい
- DynamoDBはCUだけ見ていれば良い
  - それすらもAutoScaleにある程度任せられる
- スキーマレスだけどスキーマはちゃんと決めよう
- DB自体の機能は少ない。アプリ層でがんばれ。
- 似た用途のテーブルはまとめてテーブル数を減らそう
- 性能劣化はない。気にするな。
  - ただしキーの設定は慎重に

結果、得られたもの

働かないインフラは、どれくらい働かなくて済んだのか

- フロントサーバーの運用
  - Kubernetesの機能に全乗っかり。オートスケーラーにおまかせ
- デプロイフローの運用
  - Kubernetesの機能に全乗っかり。Deploymentにおまかせ
- DBの運用
  - DynamoDBに全乗っかり。CUスパイクだけ気を付ける。
  - 性能劣化もないので本当にCUだけ見てればいいです。



手放し運転が可能に！！

データドッグのダッシュボード眺めて仕事してるフリしてればいいよ

そろそろ分析の話もさせて

Bunsekikibandatte kaetandayo.

aws SUMMIT ONLINE

Profile

今回話すひと

中村 一樹

データリード

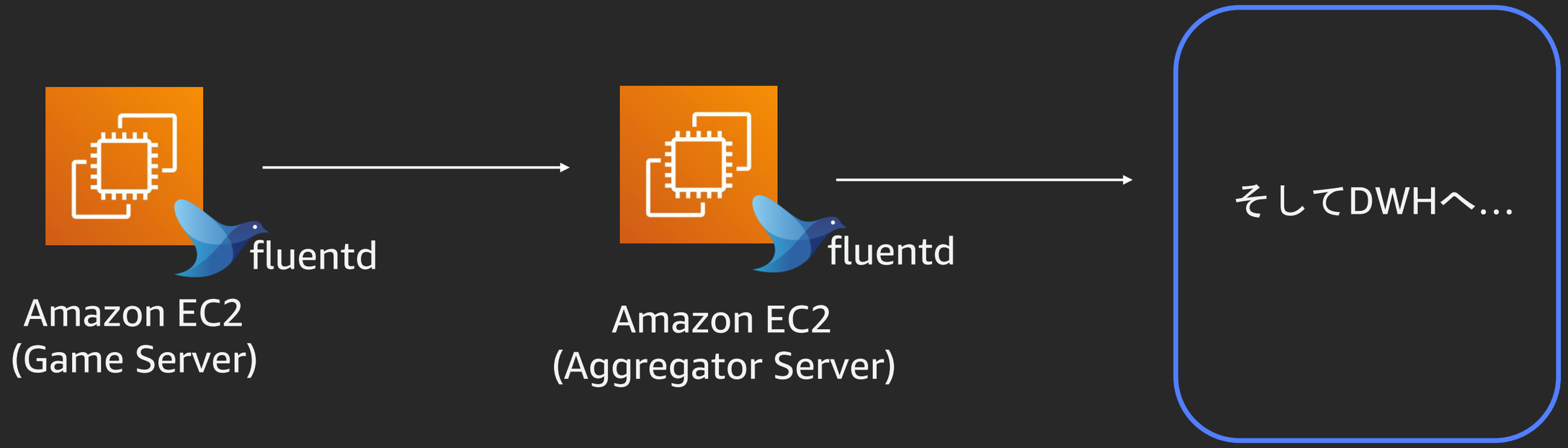
- 2008年入社、最初はコンシューマーPGだった。
- 2014年頃から突如サーバーエンジニアとなる。
- 「モンスターハンターエクスプロア」に
  初期からサーバーリーダーとして活動。
  その後クライアントリーダーなども兼任。
- 現在は機械学習や分析などデータにまつわる
  お仕事全般の責任者をしている。
  （ジョブチェンが激しい）
- なんでも屋です

今まで使っていた分析基盤

kekkou taihendattayo.

aws SUMMIT ONLINE

今までの構成図



THE　シンプル！

この構成のメリット

It’s a simple

構成がわかりやすい

構築も簡単

リアルタイム性が高い

ストリーム送信なのですぐにDWHにデータが入る

この構成のデメリット

アグリゲーターが障害点

負荷が上がり、耐えることができなくなるとログをこぼす
可能性がある

負荷に応じてインスタンスの増減をしないといけない（人力）

これを避けるためにあらかじめスペック高め
多めのサーバーが必要になる

届く保証がない

ストリームのため届く保証がない
取りこぼした場合は復旧が必要

アグリゲーターの面倒
めっちゃ見ないといけない

分析基盤だって働きたくない

ログが急激に増減しても安心・安全

穏やかな心で日々を過ごしたい・・・・・

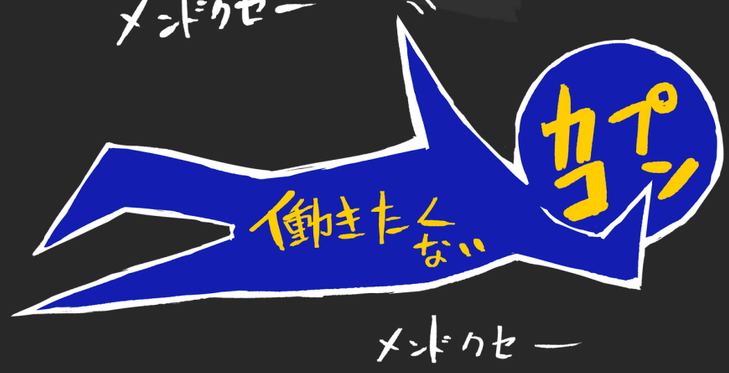
とにかくインスタンスの管理がしたくない

インスタンスのおもりはもうしたくない・・・・・

それでもコスト（お金）は抑えたい

人的コストは下げるためにお金は払うけど

それでも安いに越したことはない

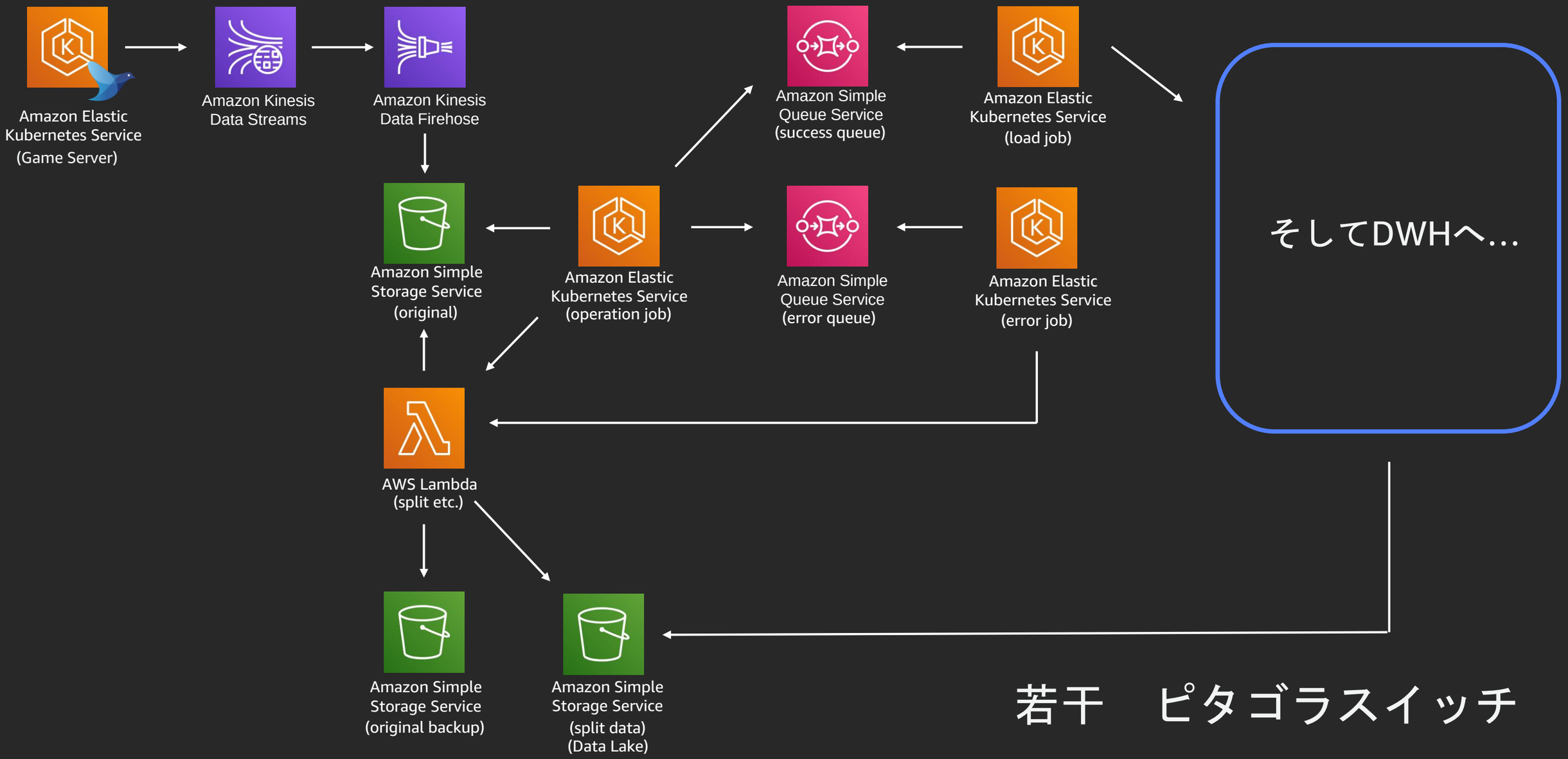


今回の分析基盤

Taihennanoha mouiyada.

aws SUMMIT ONLINE

今回の構成図



若干　ピタゴラスイッチ

Kinesis

とにかくログを溜めてくれる

設定次第で一気に来ても受け止めてくれる

重複はあるが取りこぼしはなくなる

PUSHでなくてPULLなので確実に届けられる

ログ出力の増減に関わらず一定のペースでログの処理ができる

PULLなので準備出来た状態でログを取り出せるので確実に処理できる

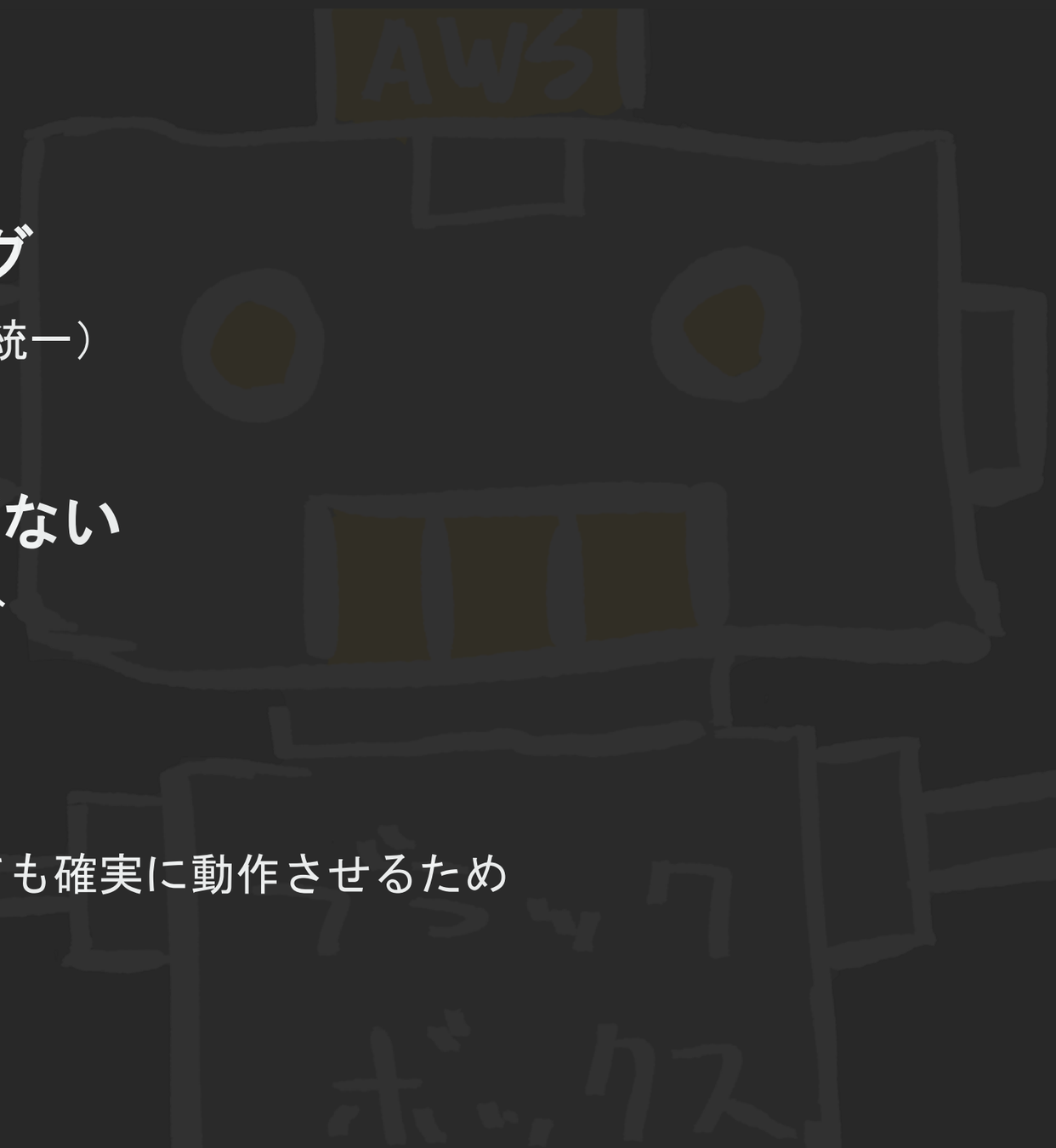
リアルタイム性は落ちるが1分1秒争うものでもないので問題ない

Kinesis

でもStreamで受けているのでデータはすべて１行・・・・・

はやくLambdaでjsonを1行ずつに改行入れるやつをデフォルトにいれてー

EKS・Lambda・SQS



分析側もEKSのjobでスケジューリング

ゲームサーバーでEKSを使っているため（技術統一）

Lambdaなのでサーバー管理の必要がない

複雑な加工等は行っていないのでLambdaで十分

SQSで動作保証

Lambdaが動かなかったり、エラー吐いたりしても確実に動作させるため

手放し運転でもほぼ問題なくなった！
でも若干複雑になったのは反省点...

実際楽になったの？

ログデータの増減に怯える日々はなくなった

急激に増えてもKinesisパイセンが受け止めてくれるから

インスタンスが調子悪いなど考える必要がなくなった

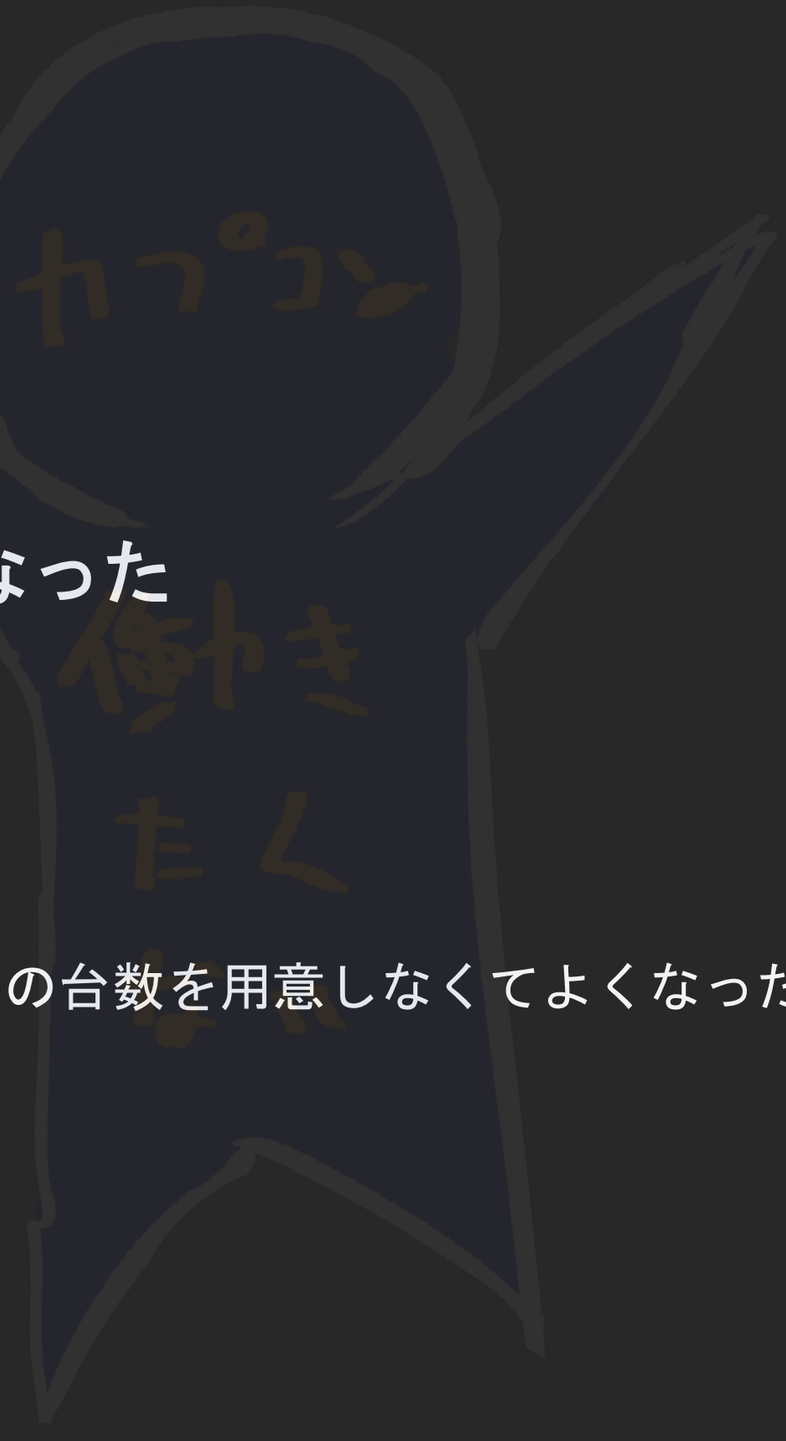
自分、サーバーレスなんで・・・・・・

常に最低限必要なコストで動いている安心感

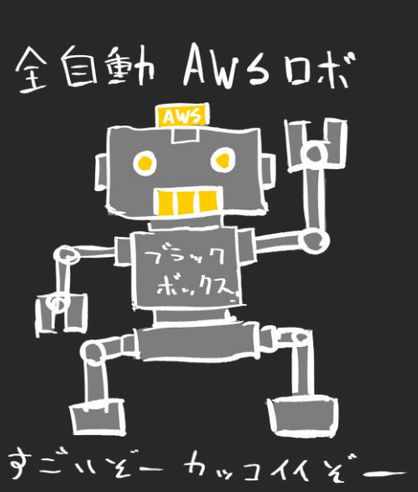
負荷を気にしてあらかじめスペックの高いインスタンス、多めの台数を用意しなくてよくなった

作業量：5%削減

精神的負荷：プライスレス



やりたくないことを
AWSに任せられるようになったｗ



Thank you!

中島 淳平
システムリード
株式会社カプコン

中村 一樹
データリード
株式会社カプコン

aws SUMMIT ONLINE

Please complete the session survey in the mobile app.

aws SUMMIT ONLINE