UNICORN

機械学習ワークロードにおける Spot&AWS Batchの活用

11/11/2020　AWS 秋のスポットインスタンス祭り

SEKI INOUE

井上 碩
@peroxyacyl

博士（情報理工学）

TOP DATA SCIENTIST @ UNICORN, Inc.
CTO @ Mist Technologies, Inc.
特任研究員 @ 東京大学

ABOUT UNICORN

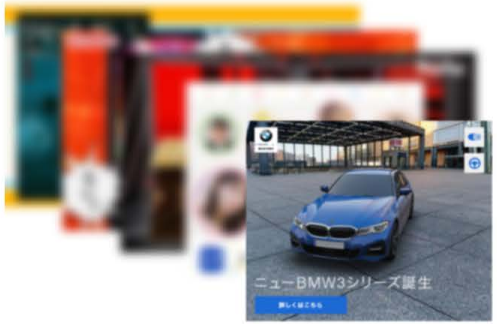
サービスのご紹介

UNICORN

デジタル広告の価値算定エンジン
＋
買付プラットフォーム

広告枠
(2300万種類/月)

×



広告在庫
(6400種類/月)

→

ユーザー群の行動が
どう変わるか?

で価値算定

広告枠の相場 = 他社の値付けは
価値算定に一切考慮しない*

業界内の立ち位置はDSPだが、買い方はかなり異なる

*買付時はオークション理論に則って最適化

UNICORN

https://unicorn.inc

UNICORN

UNICORN MODE:PERFORMANCE

MODE:PERFORMANCE

DISPLAY ADS 自動最適化エンジン

UNICORN APPLE SEARCH ADS

APPLE SEARCH ADS

APPLE SEARCH ADS 自動最適化エンジン

過去の行動を捉えるのではなく、

ユーザー群の未来の行動を予測して、

広告を広く買付する

COVERAGE - メガプラットフォームがリーチできない広告トラフィック

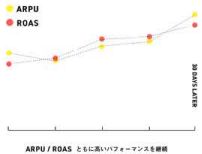
国内最大級の6,000億imp/月を用いて、Google及びSNS広告とは
異なるユーザーのシチュエーションとカバーできない全領域にてリーチ可能。

FRAUD PROTECTION - 徹底的なデータクレンジング

誤タップ誘導と不正imp&clickの検出、CTR/CVR分析、不正広告探知の専門機
関によるコンバージョンデータの検収など、クライアントの広告費用と自社の
[illegible]を保護

PERFORMANCE - 自動最適化・高ARPUとROAS

「月間6,000億impのデータ学習 + 徹底的なデータクレンジング」を基に、
最も理想的な「広告枠 x オーディエンス x クリエイティブ」の組み合わせを
予測し、ターゲットKPIに合わせて自動最適化。



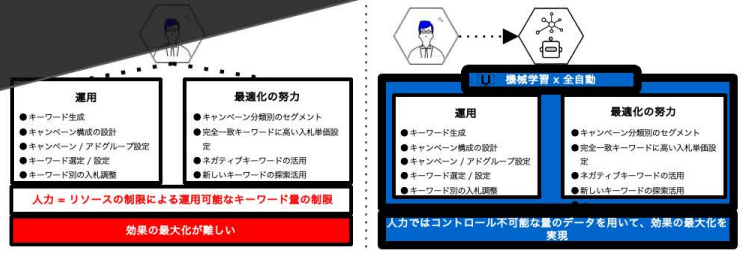
WHY - なぜApple Search Adsが重要な広告商品なのか？

- [illegible]
- 事業成長のコアに繋げる事ができる広告商品（自然流入の増加）
- ポストiOS14の環境で最も重要な広告商品

課題

解決案

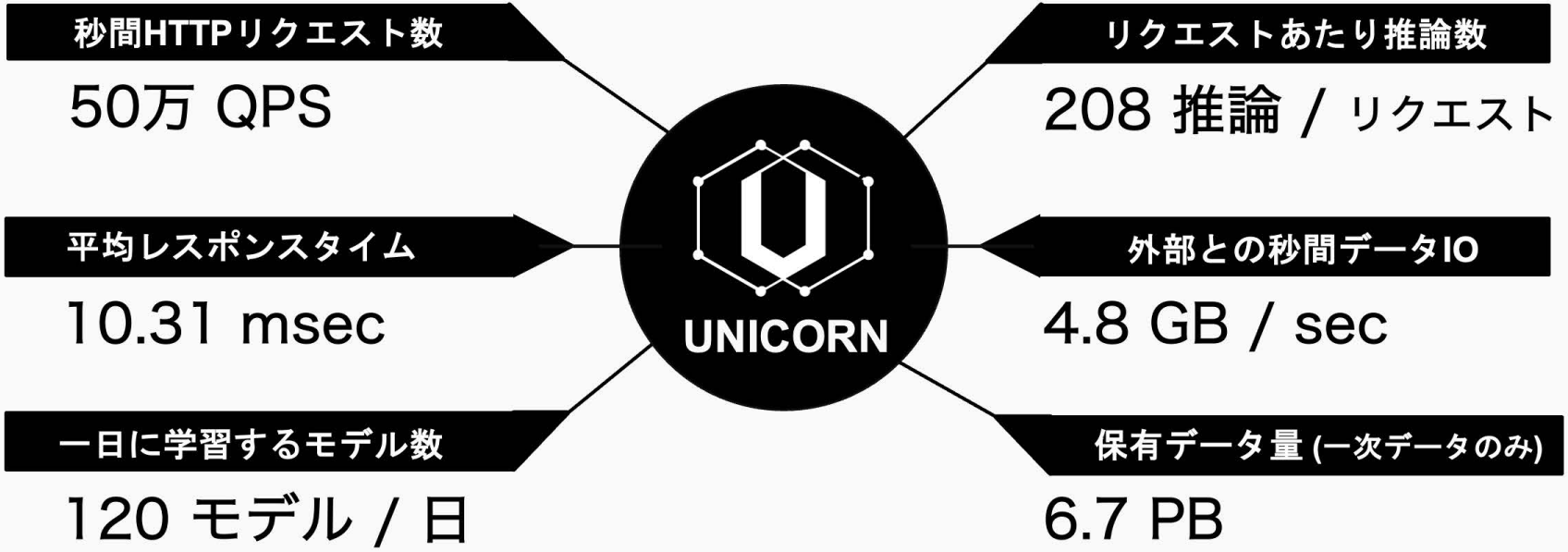
UNICORNの
Apple Search Ads自動最適化エンジン



HOW - 10万通り以上の'属性 x キーワードの組み合わせ'を自動最適化

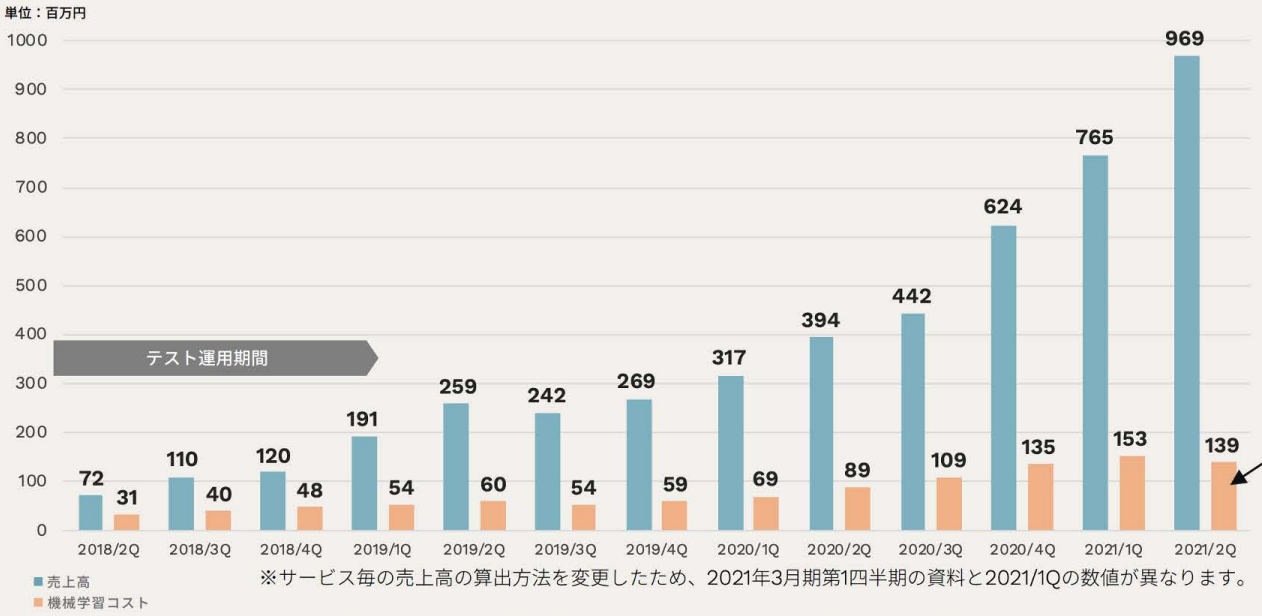
- 100+ のユーザー属性の自動生成
- 1,000+ のキーワード自動収集
- 100,000+ のユーザー属性 x キーワードの組み合わせに対する自動入札最適化

UNICORN Tech Numbers



UNICORN Biz Numbers

UNICORNの売上高／機械学習コスト推移



※サービス毎の売上高の算出方法を変更したため、2021年3月期第1四半期の資料と2021/1Qの数値が異なります。

売上高は前年同期の246%と大幅に伸長。
引き続き、機械学習も加速させ順調に精度を向上。

AWSインフラコストが大部分を占める

スポットインスタンスがなければビジネスが成立しない

※ 株式会社アドウェイズ 2021年3月期 第2四半期決算説明会資料

SPOT + UNIC

スポットインスタンスの利用状況

全体の半分以上がスポットインスタンス

ML推論・学習部分では94%

71%
Savings

$0.0198	$0.0041
Average cost per VCPU-hour	Average cost per mem(GiB)-hour

全体　：333スポット/602インスタンス

うちML関連部分

12xlarge

- ML学習　：　6スポット/6インスタンス
- ML推論　：　301スポット/319インスタンス
- DB　：　0スポット/98インスタンス

学習
- 失敗可能なタスク → Spot+Batch

推論
- オートスケール → Spot+ELB

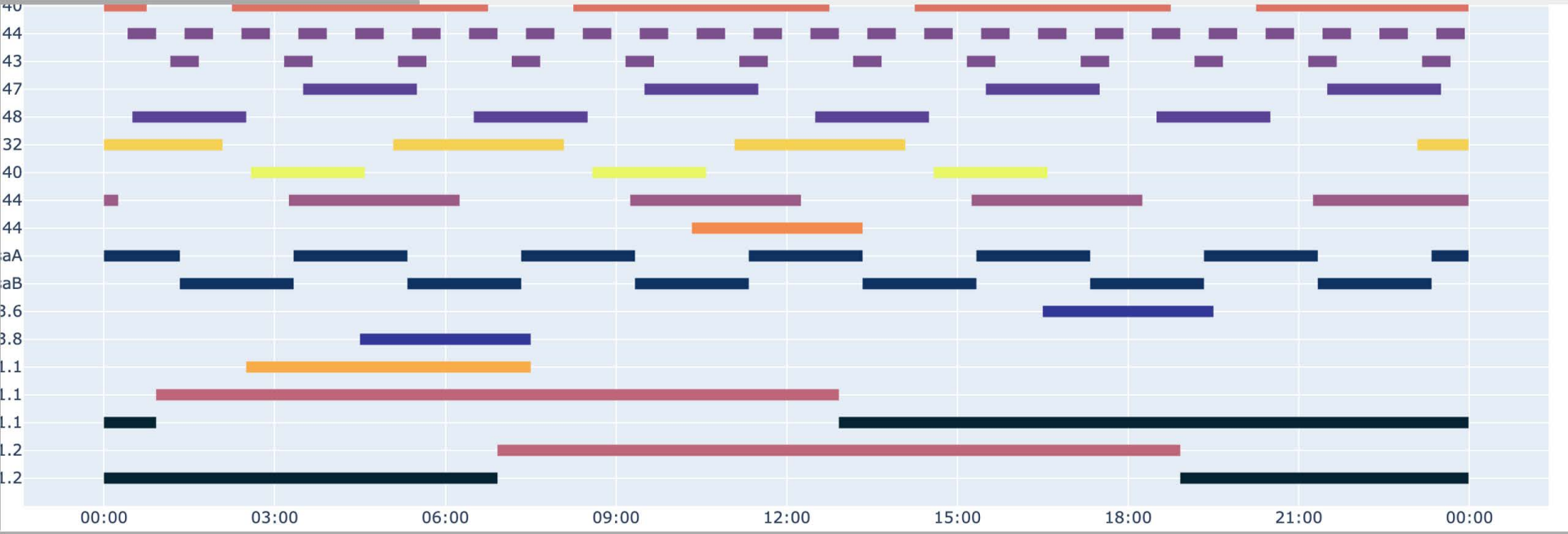
DB
- 負荷が安定 → Reserved

UNICORN モデル群 学習スケジュール

学習タスクによって
- かかる時間
- つかうCPU
- 必要なメモリ
- 必要なディスクサイズ　　が異なる

→

AWS Batchでその都度リソースを確保
第5世代 (c5, c5a, m5, r5)をSPOTで使用

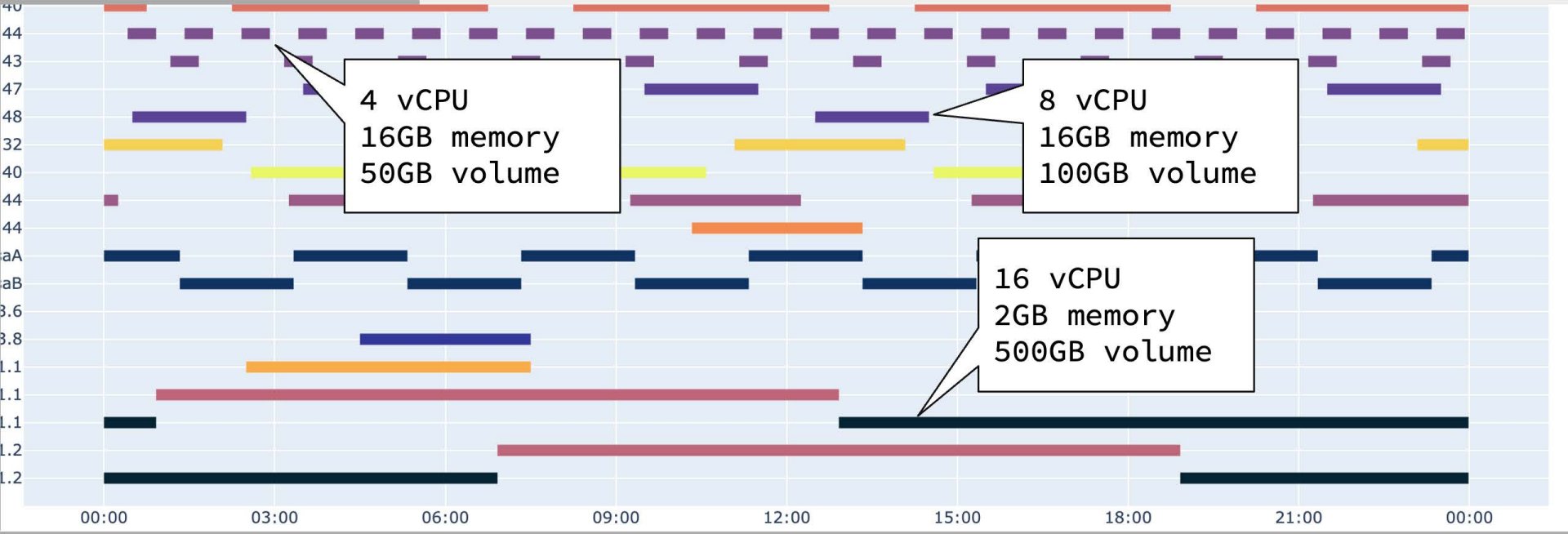


UNICORN モデル群 学習スケジュール

学習タスクによって
- かかる時間
- つかうCPU
- 必要なメモリ
- 必要なディスクサイズ　　が異なる

→

AWS Batchでその都度リソースを確保
第5世代 (c5, c5a, m5, r5)をSPOTで使用



BATCH + ML

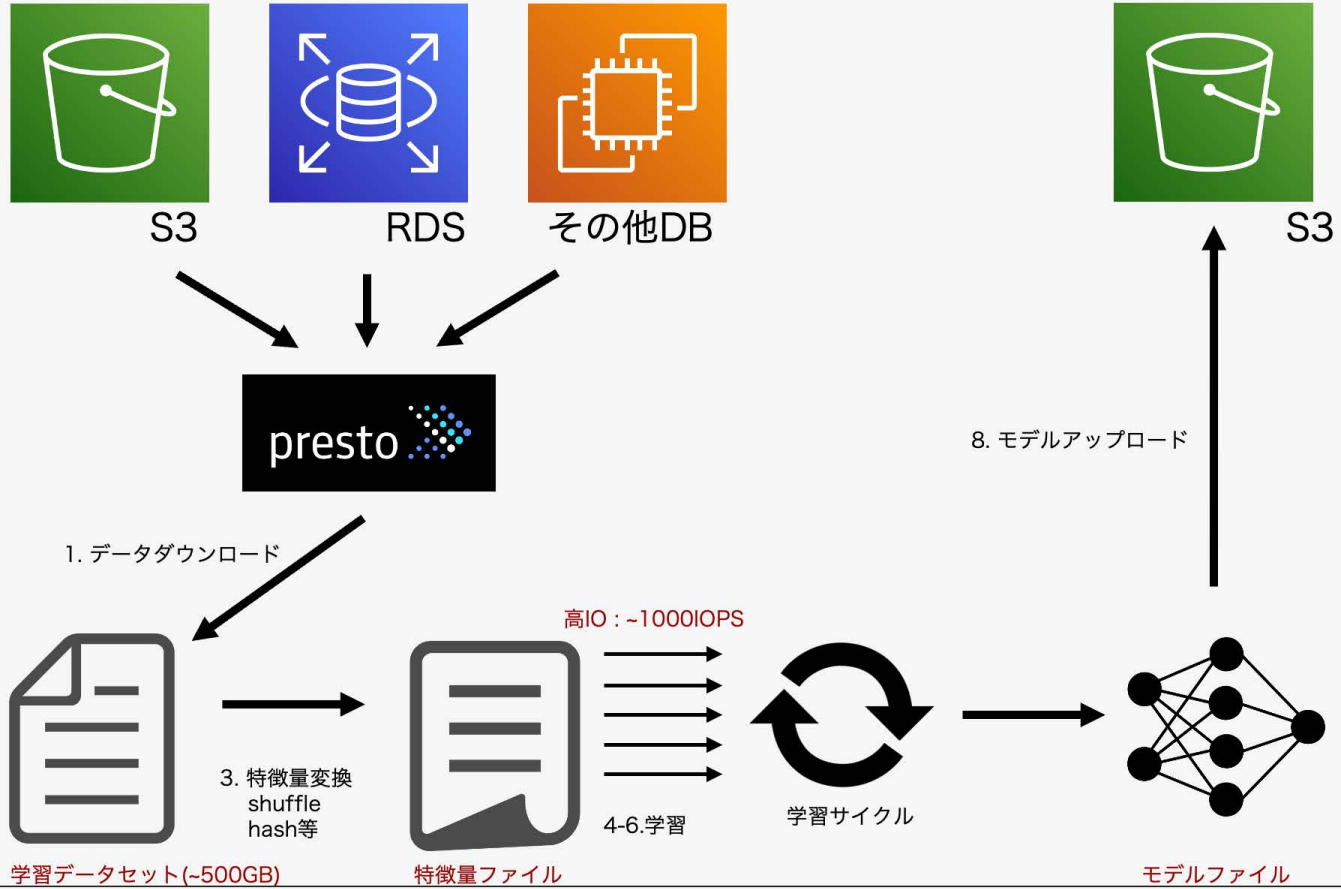
AWS Batchと機械学習

ワークロードの概要

1. データのダウンロード
2. データの検証
3. 特徴量変換

1. 学習エポック1
2. 学習エポック2
   …
1. 学習エポックN

1. モデルの検証
2. モデルのアップロード



スポットで中断しても良い学習バッチとは?

↓

失敗したときのバックアッププランを考える

失敗してもよいML学習ワークロードの例

① モデルのバージョニング

失敗した場合は
一つ前のバージョンを使う

学習のチューニング方針

- モデルバージョン間で大きな推論の差が出ないようにする
  - データセットの期間を長く取る
  - 環境変化に対する即応性とのトレードオフ

1. 学習正常終了 → S3へアップロード（S3 Versioning)
2. 推論側は、定期的にS3をポーリング

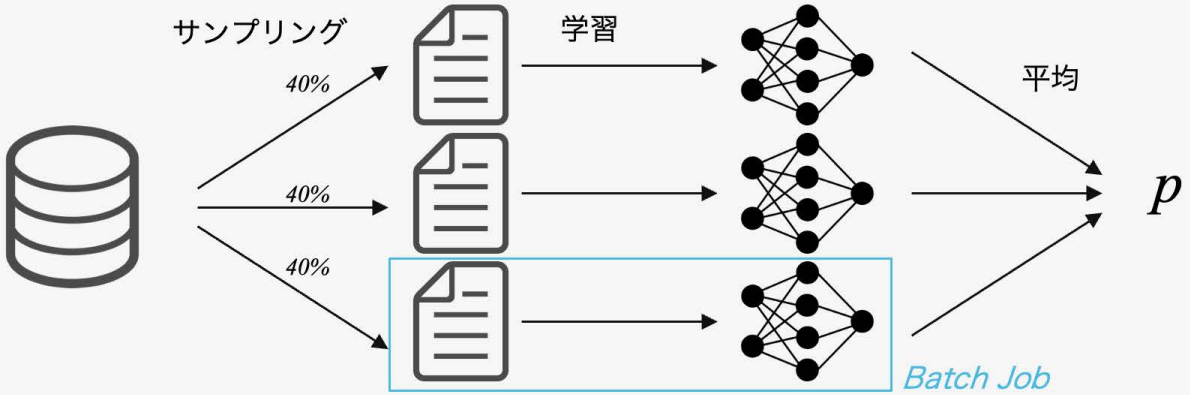
失敗してもよいML学習ワークロードの例

② マイクロモデルをアンサンブルする

学習を短く、数を多く

大きな単一のモデルで推論するのではなく、
小さなモデルを複数用意してそれらの平均を取る

学習を短く済ませ、歩留まりを上げる

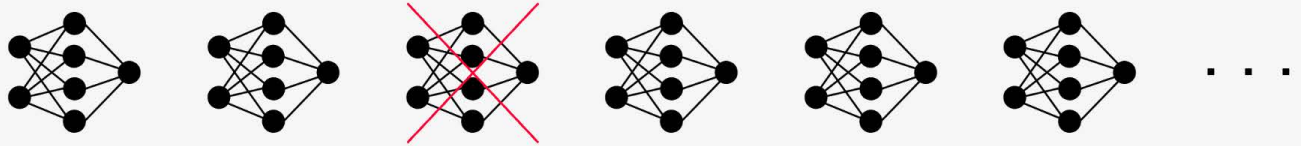


失敗してもよいML学習ワークロードの例

③ バージョニング + アンサンブル

即応性 + 安定性

最新 N バージョンを加重平均する



$$p \leftarrow 0.5p' + 0.3p'' + 0.2p'''$$

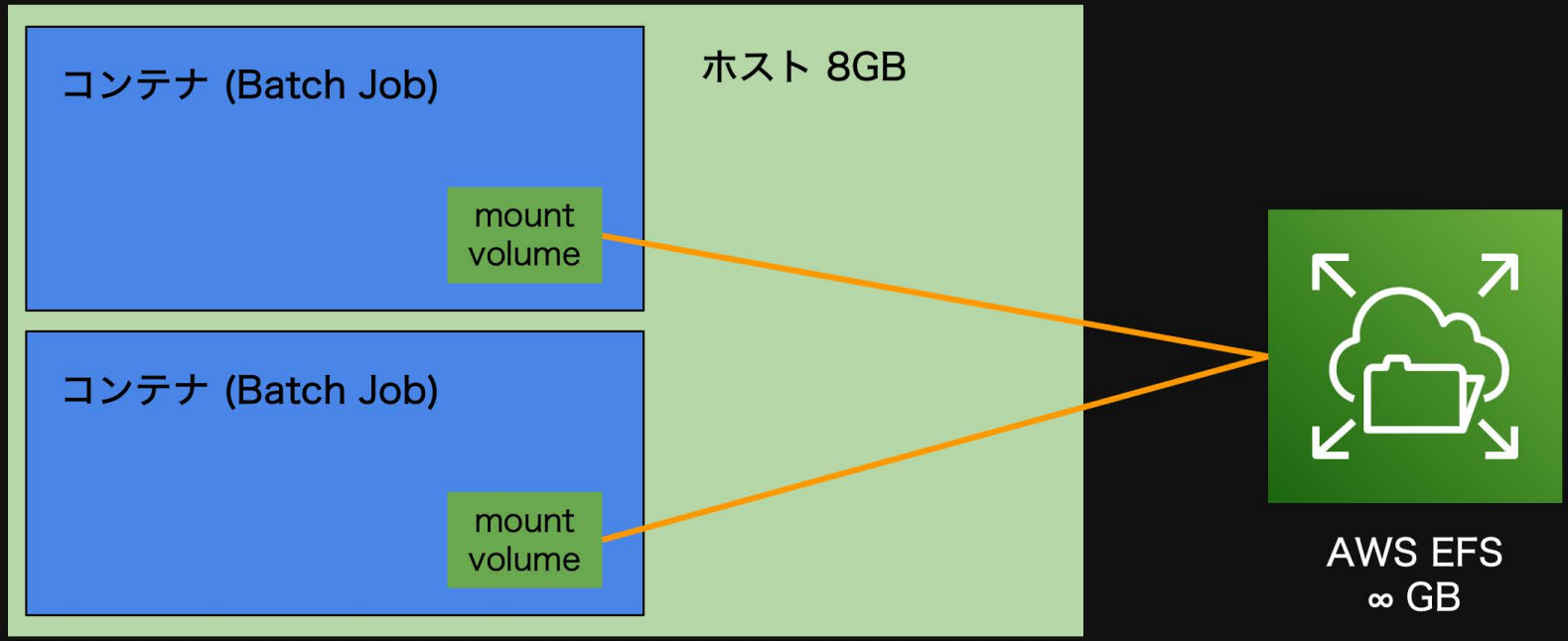
BATCH+EBS

AWS BatchでEBSを使う方法

AWS Batchの基本容量はホストあたり8GB
↓
ストレージを足す方法は３つ

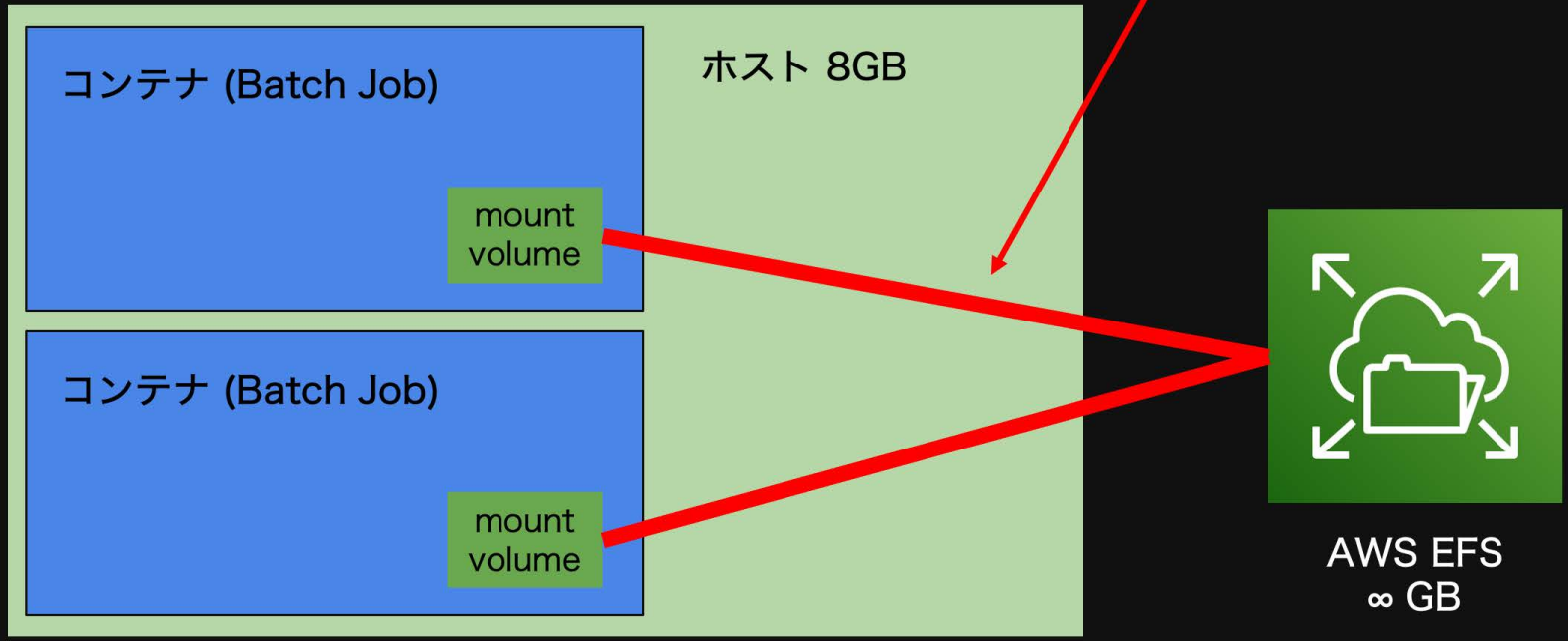
① EFSをマウントする

EFS = スケーラブルなNAS　　サイズ課金 + スループット課金



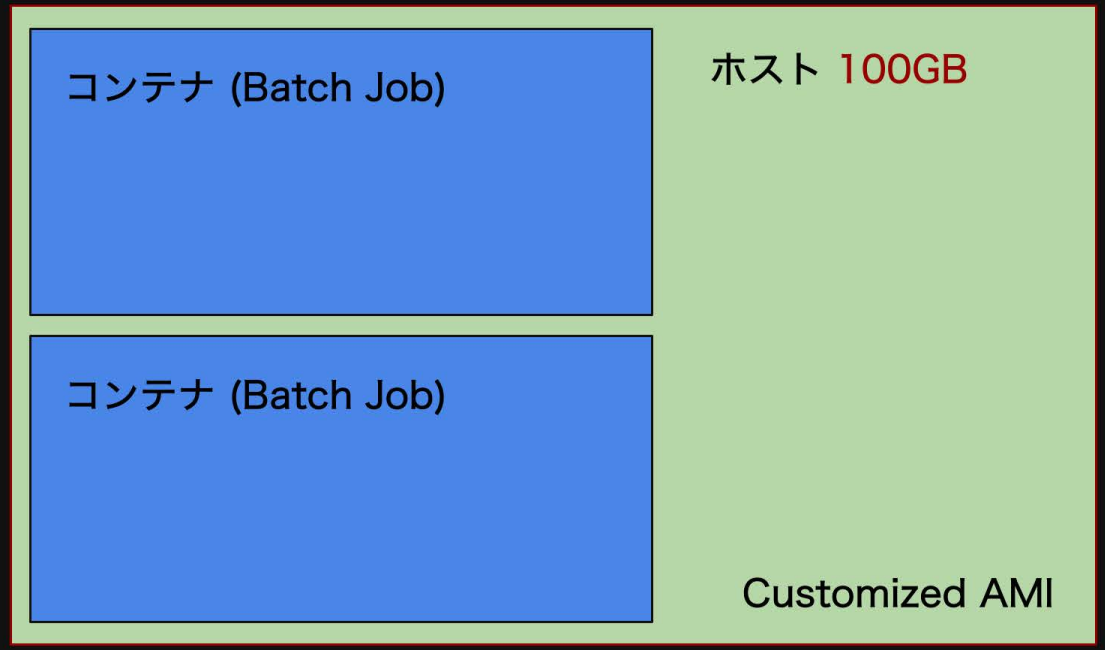
① EFSをマウントする

EFS = スケーラブルなNAS　　サイズ課金 + スループット課金



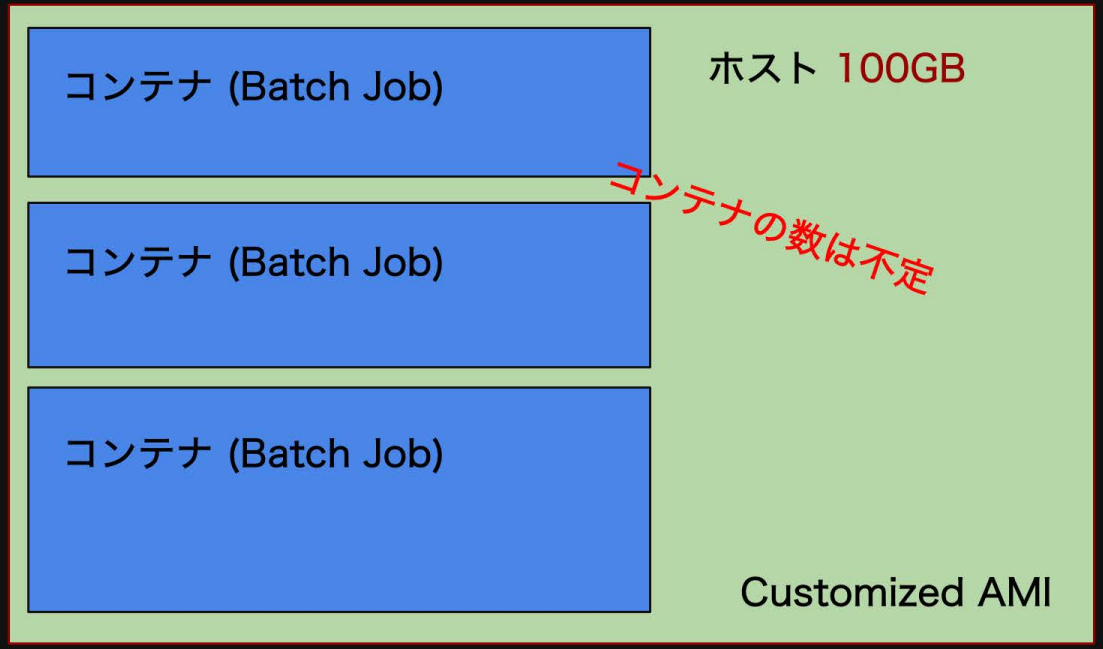
② ホストのEBSをアップグレード

ルートボリュームを大きくしたAMIを作成する



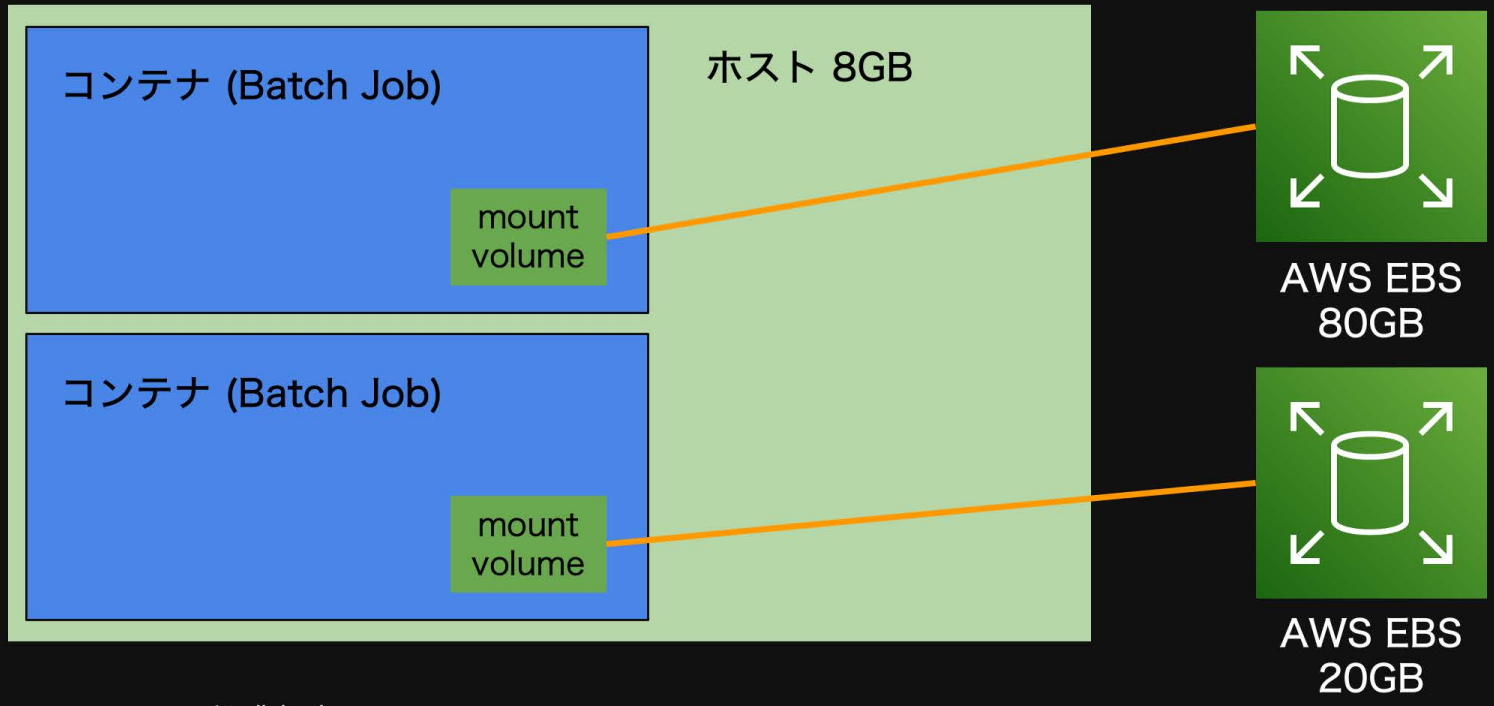
② ホストのEBSをアップグレード

ルートボリュームを大きくしたAMIを作成する



③ EBSをコンテナにマウント

コンテナのdevice socket経由でEBSをマウントする



③Batch JobごとにEBSを動的アタッチ

Jobごとにサイズを可変にして効率化

AWS Batchで標準装備のEBS

Device name	Volume size (GiB)	Attachment status	Attachment time	Encrypted	KMS key ID	Delete on termination
/dev/xvda	8	Attached	Thu Oct 22 2020 10:26:04 …	No	–	Yes
/dev/xvdcz	22	Attached	Thu Oct 22 2020 10:26:04 …	No	–	Yes
/dev/xvdg	100	Attached	Fri Oct 23 2020 03:56:28 G…	No	–	Yes
/dev/xvdh	50	Attached	Fri Oct 23 2020 19:26:20 G…	No	–	Yes
/dev/xvdi	200	Attached	Fri Oct 23 2020 20:08:23 G…	No	–	Yes

Jobごとに作成＋アタッチされた揮発領域用のEBS

Dockerコンテナでストレージを使う３つの方法

デフォルトは、インスタンスあたり8GB

	EFSをマウント	EBSをホストにマウント	EBSをコンテナにマウント
共有範囲	全Jobで共有	ホスト内で共有	Jobで専有
マウントのタイミング	Job起動時	AMI作成時	Job起動時
メリット	データが永続的 全体で共有できる	コンテナがシンプル	Job毎にサイズ/IOPSが可変 burst creditがリセット 断片化しない(st1も視野)
デメリット	スループットが高価	サイズ/IOPSの事前設定が必要 十分なリソースの確保が必要	データが完全に揮発的 消しそこねると痛い

Dockerコンテナでストレージを使う

UNICORNのワークロードでの費用感

	EFSをマウント	EBSをホストにマウント	EBSをコンテナにマウント
必要なサイズ	2,500 GB	500 GB x 6 instances	合計 2,500 GB
必要なProvisioned IO	1,500 Mbps	io2 2,000 IOPS (500 IOPS x 4 job)	gp2 (起動時のburst creditで足りたため)
月間コスト (ap-northeast)	$10,800 / month	$1,314 / month	$300 / month

③ EBSをコンテナにマウント

【EBSをdeviceとしてアクセスする】

※ 要privilegeフラグ

Dockerコンテナ起動時のENTRYPOINTで以下の処理を行う

① awscliでEBSを作成

```
$ aws ec2 create-volume
```

② 使用可能なデバイスファイルを探す (例: /dev/xvdf)

```
$ [while文などで]
```

③ 作ったEBSをホストインスタンスにアタッチ

```
$ aws ec2 attach-volume
```

④ デバイスのファイルシステムをフォーマット

```
$ mkfs.xfs -f /dev/$DEVNAME
```

⑤ マウント

```
$ mount /dev/$DEVNAME $EBS_PATH
```

⑥ EBS削除の設定 (後述)

https://github.com/peroxyacyl/batch.tmpl

EBSの削除

EBSが残ると課金が続く

↓

確実な削除が必要

lambdaで監視するのも有効

EBS削除のタイミングと方針

1. 正常終了

ENTRYPOINTでコマンド後に実行

2.プログラムの異常終了

ENTRYPOINTでtrapする

```
function rmebs {
    set +e
    umount -l /media/data
    aws ec2 detach-volume --force --volume-id $VOLUME_ID
    timeout 600 bash -c "waitebsstate $VOLUME_ID available"
    aws ec2 delete-volume --volume-id $VOLUME_ID && echo "volume successfully deleted"
}
trap rmebs EXIT
mkebs

$@
```

https://github.com/peroxyacyl/batch.tmpl

EBSの削除

EBSが残ると課金が続く

↓

確実な削除が必要

lambdaで監視するのも有効

EBS削除のタイミングと方針

3. スポットインスタンスの回収
4. ジョブの手動キャンセル

中断通知内のdetachが間に合わない
キャンセルの場合は即時に落ちる
→ Delete On Termination機能を使う

```
echo "enable DeletionOnTermination to ensure eventual deletion"
cat << EOF >> mapping.json
[
  {
    "DeviceName": "/dev/${DEVNAME}",
    "Ebs": {
      "DeleteOnTermination": true
    }
  }
]
EOF
aws ec2 modify-instance-attribute --instance-id $INSTANCE_ID --block-device-mappings file://mapping.json
```

https://github.com/peroxyacyl/batch.tmpl

Tips

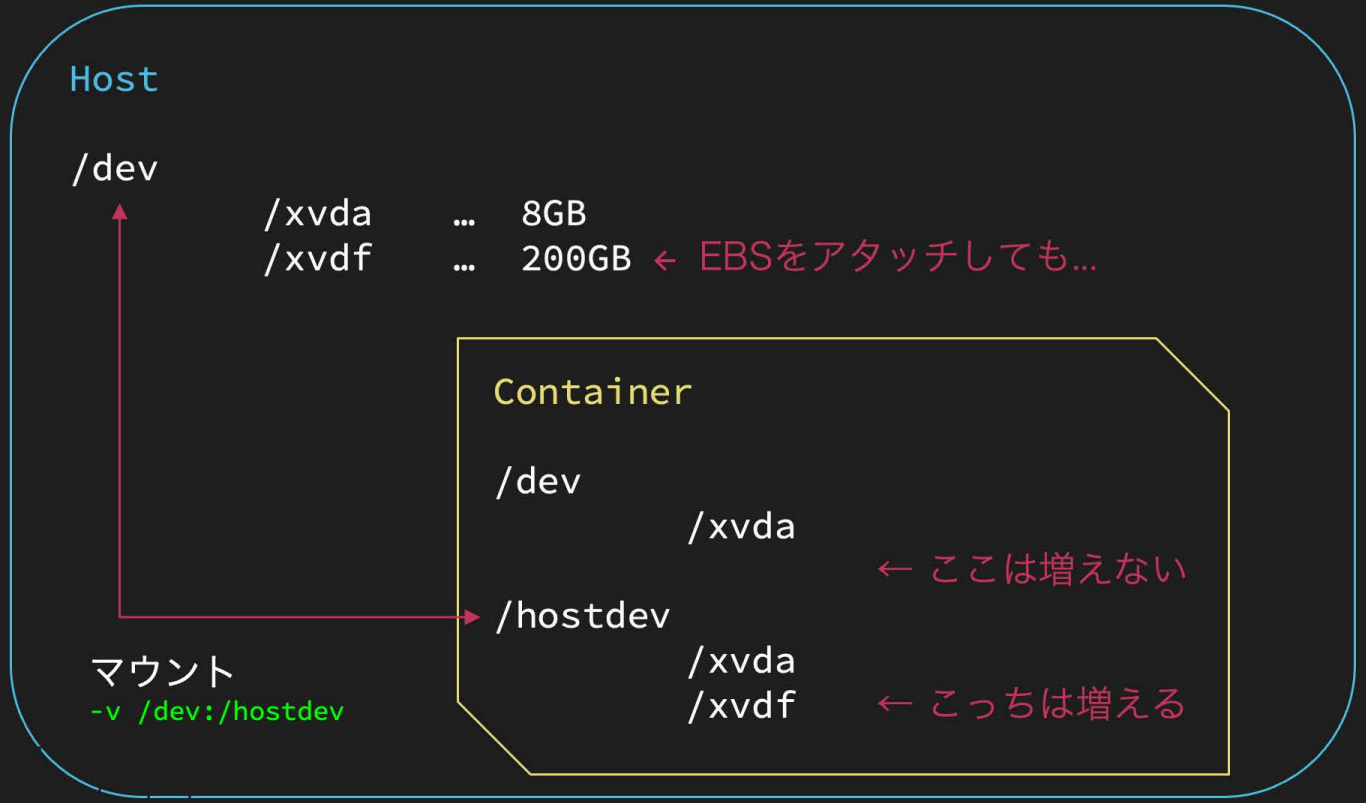
/devをコンテナにマウント

コンテナ内の/devは
デバイス追加時に更新されない

↓

/devを仮想ボリュームとして
コンテナにマウントする

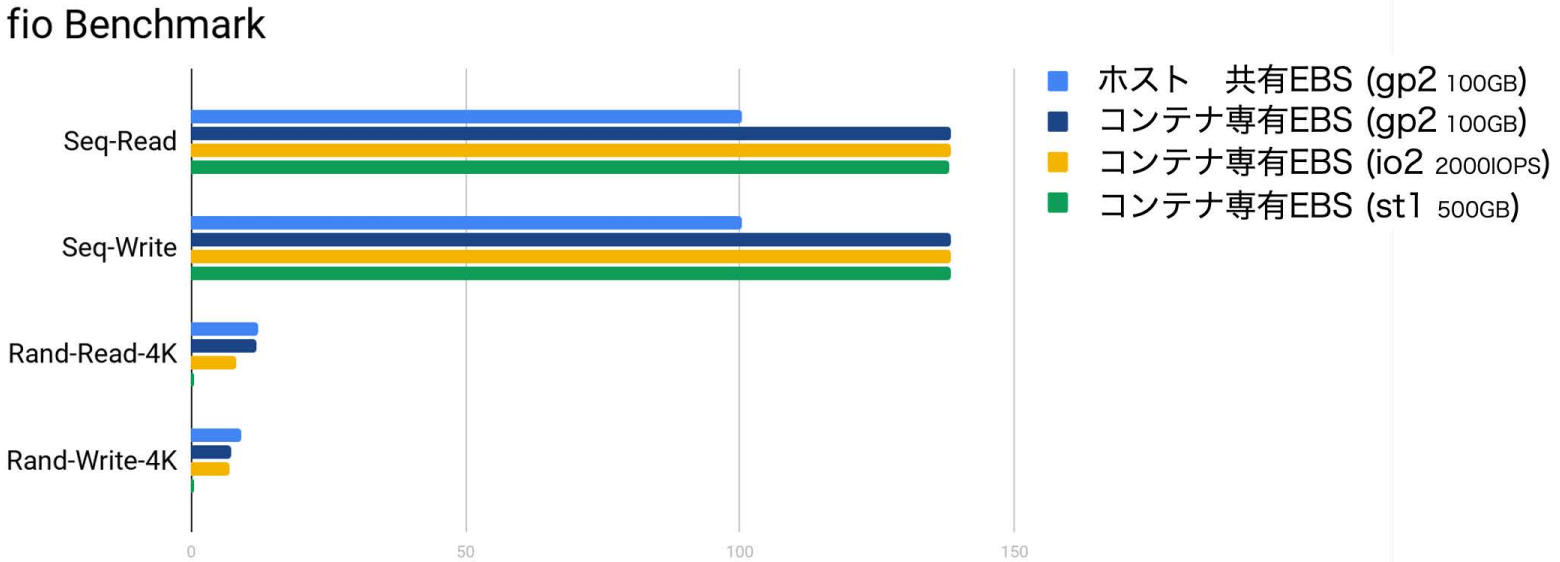
第三者のdocker imageには注意



https://github.com/peroxyacyl/batch.tmpl

EBSベンチマーク

コンテナ直接アタッチのパフォーマンスは問題なし



https://github.com/peroxyacyl/batch.tmpl

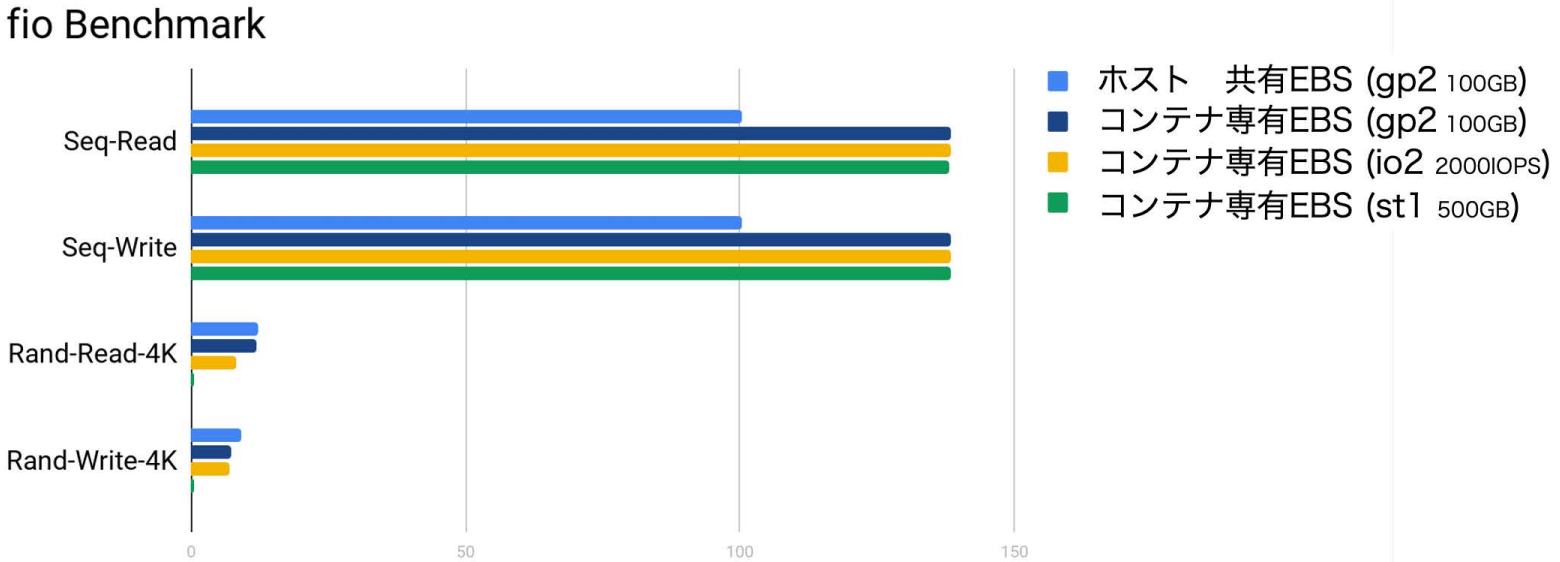
Dockerコンテナでストレージを使う

UNICORNのワークロードでの費用感

	EFSをマウント	EBSをホストにマウント	EBSをコンテナにマウント
必要なサイズ	2,500 GB	500 GB x 6 instances	合計 2,500 GB
必要なProvisioned IO	1,500 Mbps	io2 2,000 IOPS (500 IOPS x 4 job)	gp2 (起動時のburst creditで足りたため)
月間コスト (ap-northeast)	$10,800 / month	$1,314 / month	$300 / month

EBSベンチマーク

コンテナ直接アタッチのパフォーマンスは問題なし



https://github.com/peroxyacyl/batch.tmpl

開発でもBatchを使う

チューニングなどの高コストな計算をクラウドにオフロード

```
$ ./runbatch.sh --cpu 4 --memory 16000 --volume 100 "python optuna.py"
```

このコマンドは下記を行う

- ディレクトリをdockerに固めてECRにpush
- job definitionの発行
- 4 vCPUs, 16GB メモリ, 100GB EBSを確保
- batch上で `python optuna.py` を起動
- Log StreamのURLを出力

https://github.com/peroxyacyl/batch.tmpl

まとめ

1. 弊社ではスポットインスタンスでコストをオンデマンドより 71% 下げている
2. 学習ワークロードでは、 失敗したときのバックアッププラン を作る
    a. アンサンブル学習 が歩留まりを上げるために有効な一例

1. AWS BatchはJobごとに異なる負荷を持つときに効率的
    ...だが, 作業用ストレージ領域が足りない
1. EBSをコンテナに直接マウント する方式が有効

2. 開発やチューニングにもAWS Batchは使える

https://github.com/peroxyacyl/batch.tmpl

We are hiring

JS SDK

自然言語処理

Data ETL

Data Visualization

Rich Creative Design

App SDK

SRE

強化学習

Microservice Orch.

Tech. Account Managing

golang
ruby
rust
assembly
TypeScript
SQL

WE ARE HIRING

UNICORN

contact@unicorn.inc