CUS-17

高度運転支援向け画像認識のための Machine Learning Production Line

新原　竜馬
株式会社デンソー　担当係長

Agenda

1. 会社紹介　および　AD&ADAS事業紹介
2. 画像センサの概要
3. 機械学習プロセスと課題、学習環境への要件
4. アマゾン ウェブ サービス（AWS）での解決策
5. まとめと今後

1

会社紹介　および　AD&ADAS事業紹介

社名	**株式会社デンソー**
設立	1949年12月16日
本社所在地	〒448-8661　愛知県刈谷市昭和町1-1
資本金	1,875 億円
売上収益	連結 5兆1,535 億円　　* 2019年4月1日～2020年3月31日
営業利益	連結 611 億円　　* 2019年4月1日～2020年3月31日
当期利益* *親会社の所有者に帰属	連結 681 億円　　* 2019年4月1日～2020年3月31日
従業員数	連結 170,932人　　単独 45,280人
連結子会社数	200 社　　(日本 64、北米 23、欧州 32、アジア 74、その他 7)
持分法適用関連会社数	88社　　(日本 24、北米11、欧州 17、アジア32、その他 4)

デンソーの事業

注力４分野

デンソーは新しいモビリティの価値を提供するとともに、FAや農業の工業化に取り組んで、
社会・産業界の生産性向上に貢献します。

電動化

環境負荷の低減と
高効率な
移動の実現

ＡＤ＆ＡＤＡＳ事業

先進安全/自動運転

交通事故のない
安全な社会と快適で
自由な移動の実現

コネクティッド

クルマ・ヒト・モノが
つながる新たな
モビリティ社会の実現

非車載事業（FA/農業）

社会・産業界の
生産性向上に
貢献

AD&ADAS　アプリケーション

車間距離制御システム
(Adaptive Cruise Control)

レーンキープアシスト

駐車支援

ドライバステータスモニタ

出会頭衝突警報

衝突回避ブレーキ

歩行者衝突回避ブレーキ

エアバッグ

ポップアップフード

緊急通報システム

2 画像センサの概要

高度運転支援は　認識 ・判断・操作　の一部を代行するシステム

高度運転支援の"認知"を支えるデンソーの製品例

画像センサ（**数M pixel**）

RADAR

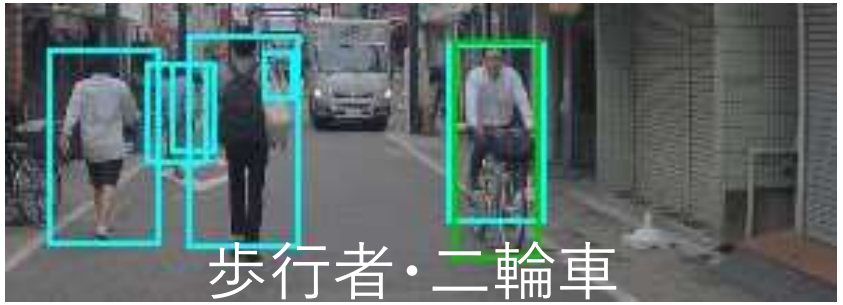
引用: DENSO Official Channel (YouTube)

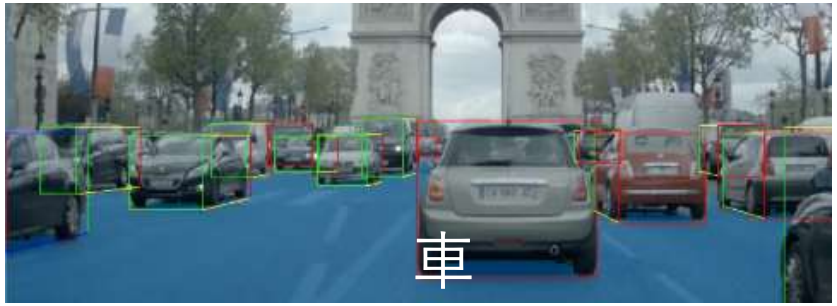
画像センサの特徴　多様なシーン　（地域 / 天候 / 時間帯）

地域性 / 天候 / 時間帯によらず認識しなければならない

画像センサの特徴　多機能

多様なものを認識しなくてはならない



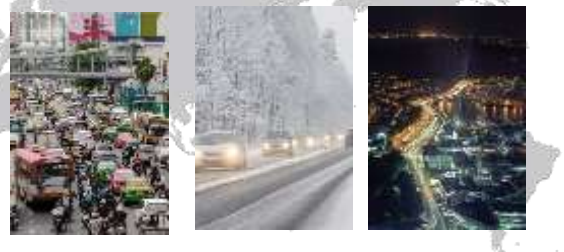


画像センサ特徴

画像センサ特徴

情報量が多い(数 Mpixel)

多様なシーン（地域/天候/時間帯）



多機能

開発の特徴

特徴を記述しきれない
⇒機械学習が得意な領域
主にCNN(Convolutional Neural Network)を活用

データ量が多い
数M pixels、数十万キロ以上のデータ

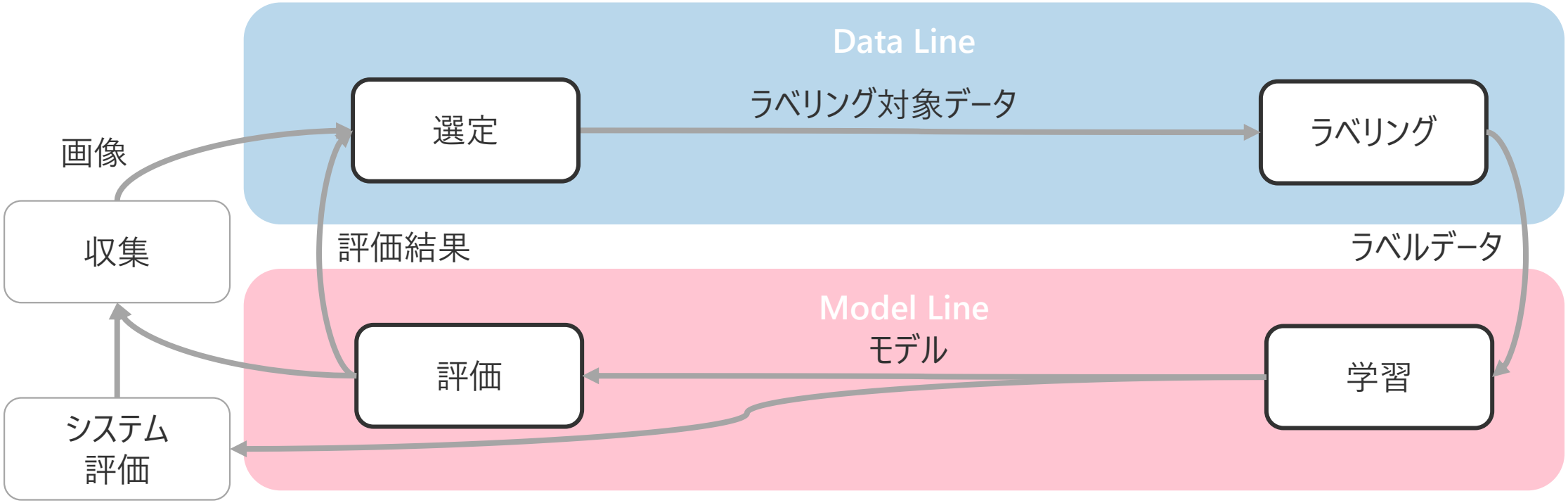
認識対象が様々
⇒多種の機械学習モデルが必要

多量（画素数、シーンのバリエーション）なデータを多様な機械学習を用いて開発

3

機械学習プロセスと課題、学習環境への要件

機械学習プロセス概要



上記のプロセスを高品質な製品を製造する生産ラインのように機械学習モデルを作る、
という思いから **"Machine Learning Production Line(MLPL)"** として整備を進めている

Data Line （選定、ラベリング） / Model Line（学習、評価）のループ

現行プロセスの課題

AI業界の抱える課題

⑨ 透明性の原則
AI サービスプロバイダ及びビジネス利用者は、AI システム又は AI サービスの入出力等の検証可能性及び判断結果の説明可能性に留意する。

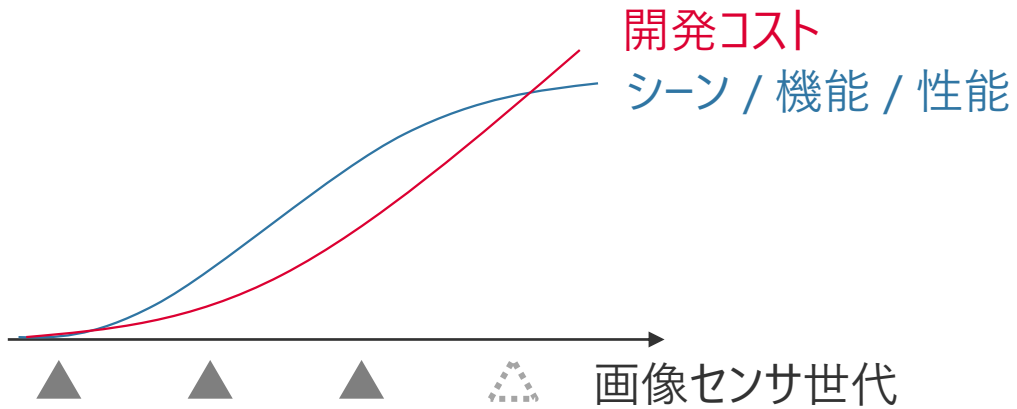
経産省 ＡＩ利活用ガイドラインより引用

- **説明可能性**のベースとなるのは、
  入力、環境、パラメタ、出力などの
  AIのトレーサビリティ(トレサビ)
  - データの素性、学習パラメタなどの情報

トレサビ確保の統一的な仕組みが必要

開発課題

- 画像センサの世代ごとに
  シーン(地域性/天候/時間帯)、機能、性能は上昇、
  開発コストも増加



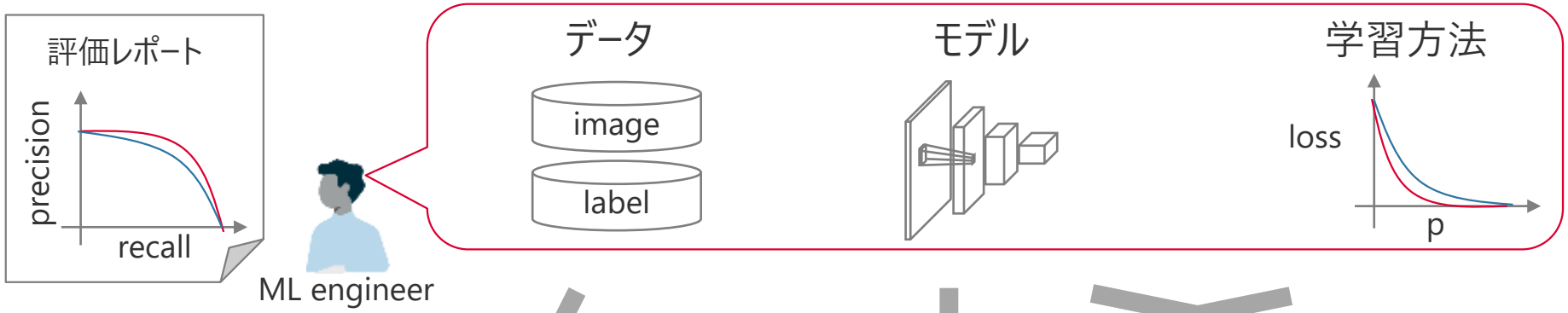
生産性を向上し続ける

トレサビ確保と生産性向上を両立した機能をMLPLで実現することが必要

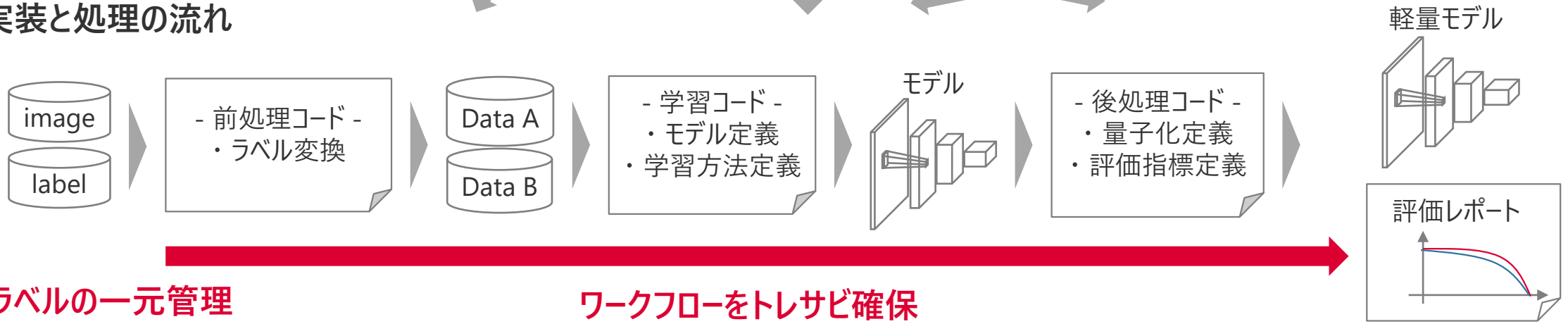
トレサビ確保のための MLPL への要件

学習プロセス（CNNモデル開発）の詳細

・主に性能向上するには、 データ / モデル / 学習方法 の3つからアプローチ



実装と処理の流れ

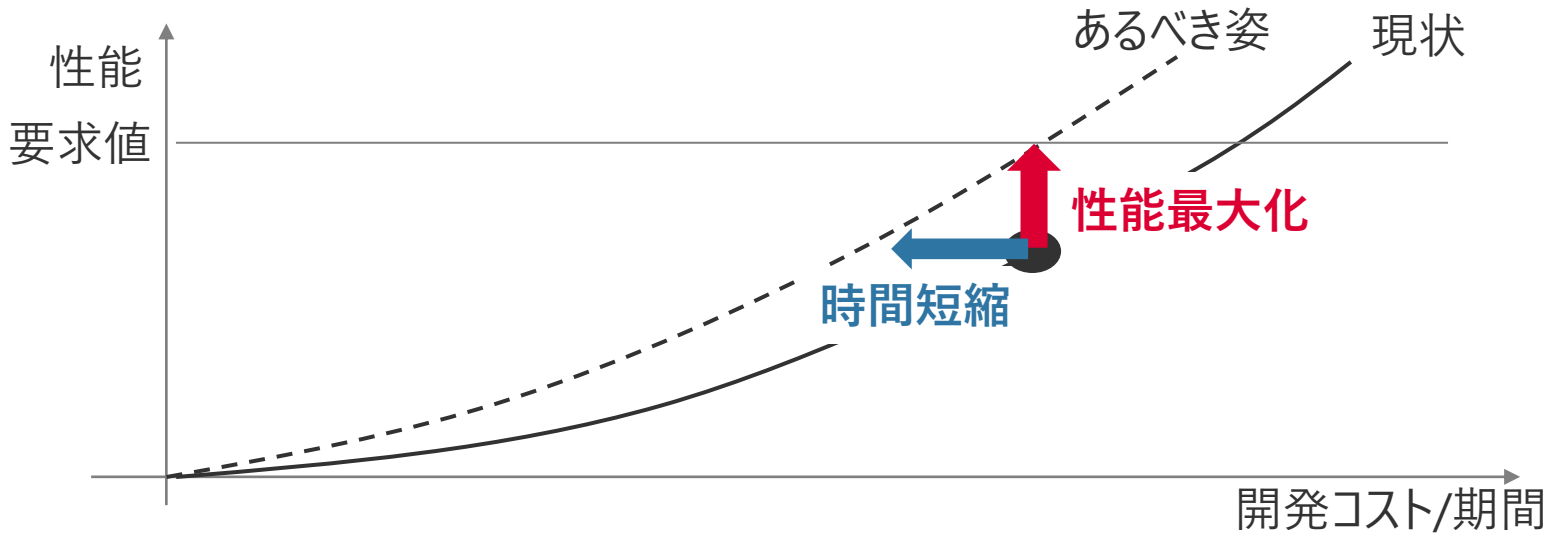


データとラベルの一元管理

ワークフローをトレサビ確保

要件① データとラベルの一元管理、要件②ワークフローをトレサビとして確保する

生産性向上の考え方と MLPLへの要件 の整理



性能最大化 / 時間短縮の軸について、Data Line / Model Lineの4象限で学習基盤としての要件を整理

	性能最大化	時間短縮
Data Line（選定・ラベリング）	生産性要件①	生産性要件②
Model Line（学習・評価）		生産性要件③

Data Line / Model Line、性能最大化 / 時間短縮の両軸で要件を整理する

性能最大化 のための MLPL への要件（生産性要件①）

機械学習とデータの前提

- 性能が高いのは、ラベルありデータを用いる教師ありデータ
- ラベリングコストは安くはないので、　ラベルありデータ　＜＜　ラベルなしデータという状態

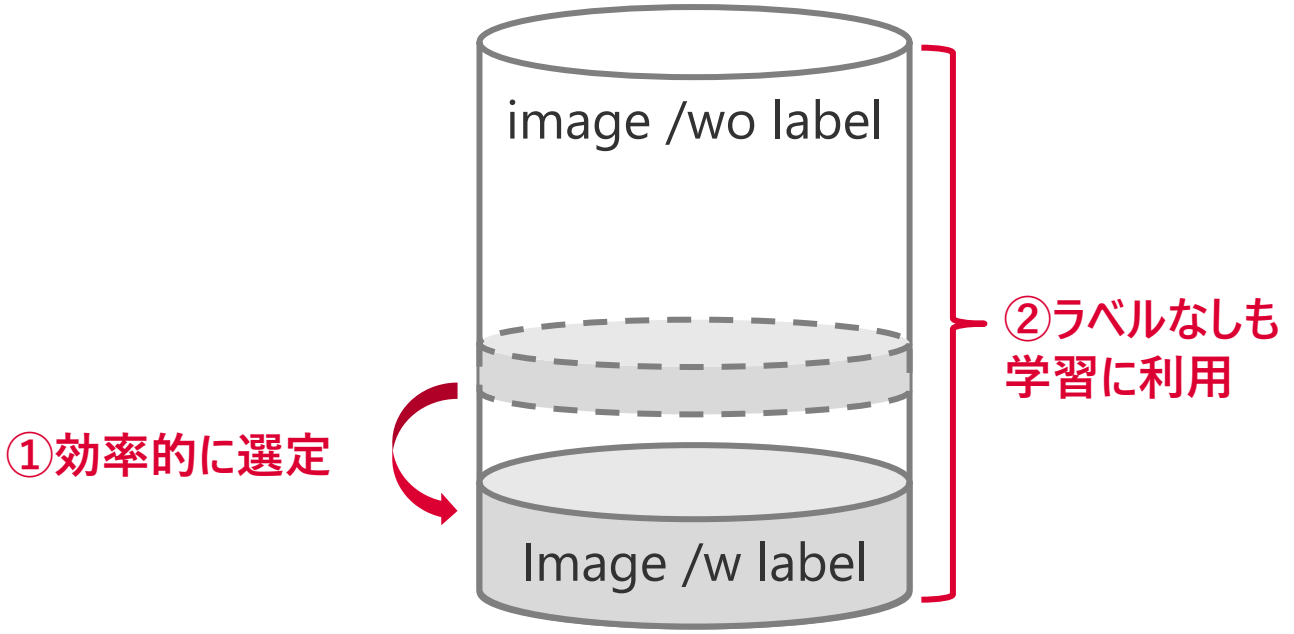
アプローチ

①Data Line：ラベリング対象を効率的に選定

- 実現技術例
  - 画像検索
  - 能動学習

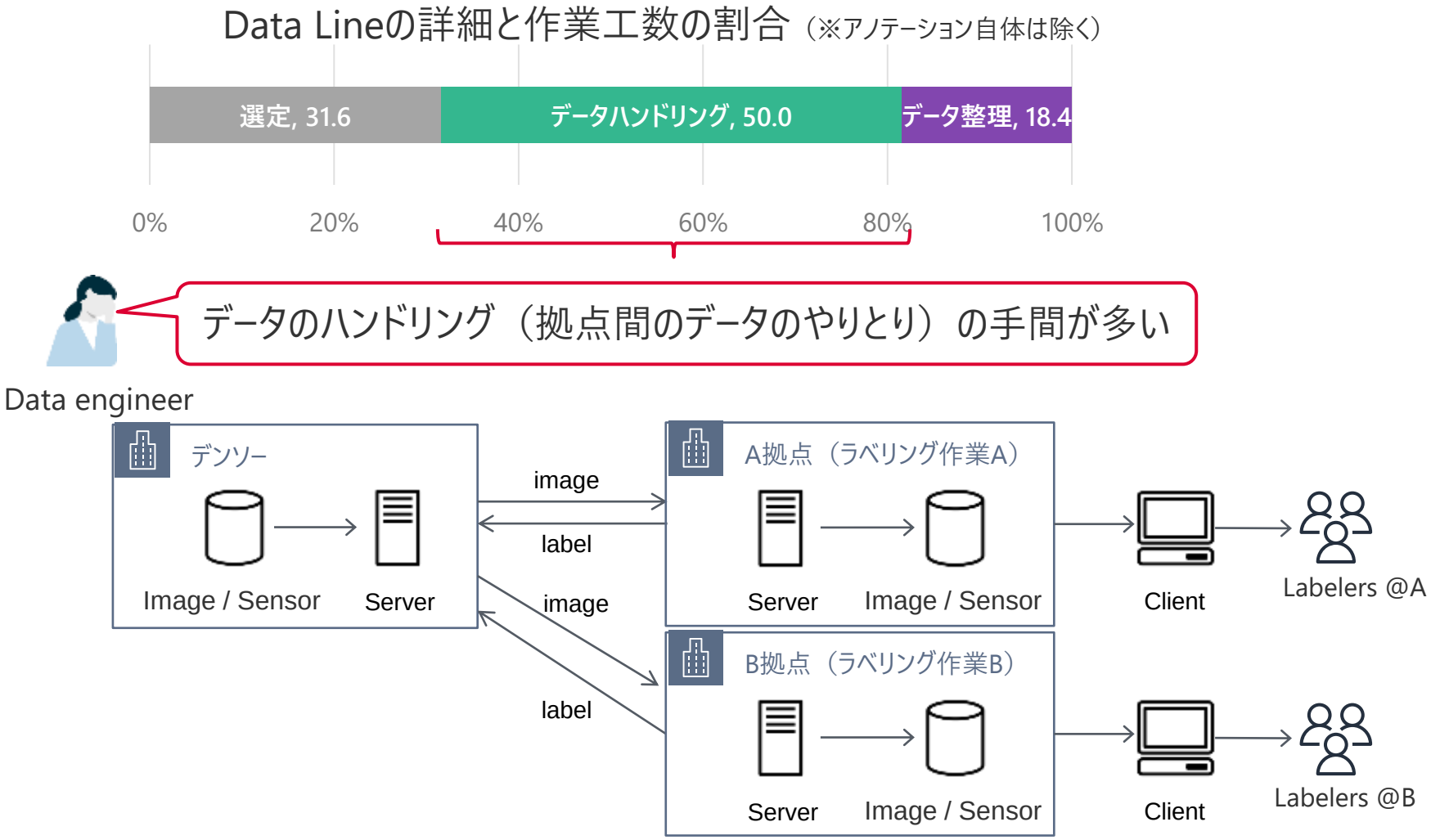
②Model Line：ラベルなしデータ学習に利用

- 実現技術例
  - 教師なし学習
  - 半教師あり学習



データとラベルの対応付けが管理されている＝データとラベルの一元管理

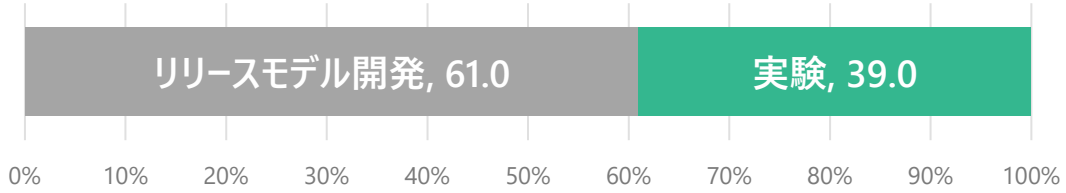
時間短縮 × Data Line の MLPL への要件 (生産性要件②)



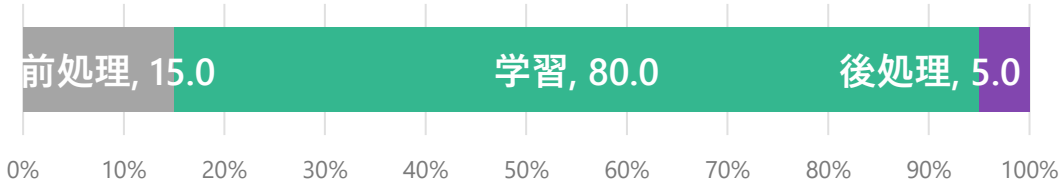
データのハンドリング効率化が要件

時間短縮 × Model Line の MLPL への要件 (生産性要件③)

Model Lineの詳細と作業工数の割合



モデル開発 1 つにかかる計算リソースの内訳



・もっと実験する時間を増やしたい
・リリースで繰り返し作業をするのが非常に手間

ML engineer

・学習をもっと高速化したい
（分散学習、最新のHWを使いたい）

繰り返し作業が不要な環境の構築⇒ワークフローの自動化

強力なHWリソースを確保し続ける
（クラスタ形成可能、最新HWの確保）

ワークフローの自動化、強力なHWリソースが要件

MLPLの要件と目標

トレサビ確保の要件

プロセス	要件
Data Line	トレサビ①データとラベルの一元管理
Model Line	トレサビ②ワークフローのトレサビを確保

生産性の要件

プロセス	性能最大化	時間短縮
Data Line	生産性①データとラベルの一元管理 （能動学習 / 画像検索）	生産性②データのハンドリング効率化
Model Line	生産性①データとラベルの一元管理 （半教師あり/教師なし学習）	生産性③強力なHWリソース 生産性④ワークフロー自動化

MLPL のポイント、要件と目標

MLPLのポイント	要件	目標
強力なHWリソース	生産性③強力なHWリソース	オンプレミス/AWSの実現可能性、 対比コストが安くなること
データ管理基盤	生産性①データとラベルの一元管理 生産性②データのハンドリング効率化	現行プロセス比 60%以下にする(※1)
ワークフロー自動化	生産性④ワークフロー自動化	現行プロセス比 70%以下にする(※1)
トレサビ確保	トレサビ①データとラベルの一元管理 トレサビ②ワークフローのトレサビを確保	トレサビ確保を可能とする

※1. 目標値現行プロセス比で今後の開発を見据えて算出

4

アマゾン ウェブ サービス（AWS）での解決策

MLPL 開発手順

MLPL のポイント、要件と目標　※再掲

MLPLのポイント	要件との対応付け	目標
強力なHWリソース	生産性③強力なHWリソース	オンプレミス/AWSの実現可能性、 対比コストが安くなること
データ管理基盤	トレサビ①データとラベルの一元管理 生産性①データとラベルの一元管理 生産性②データのハンドリング効率確保	現行プロセス比 60%以下にする(※1)
ワークフロー自動化	③ワークフロー自動化	現行プロセス比 70%以下にする(※1)
トレサビ確保	②ワークフローのトレサビを確保	トレサビ確保を可能とする

手順

1. オンプレミス v.s. AWSの比較
   - 「強力なHWリソース」を確保するためのHW要件を満たせるか確認し、コストを比較する

2. AWSでのMLPL 構築
   - 「データ管理基盤」、「ワークフロー自動化」、「トレサビ確保」を実現する開発基盤を構築する

強力なHWリソース

基盤選定方針
- AWSでも強力なHW(GPU / 高速ストレージなど）が使用可能
- オンプレミス v.s. AWSの比較するために、クラウドエコノミクス / Amazon SageMaker TCO Calculatorを用いて算出

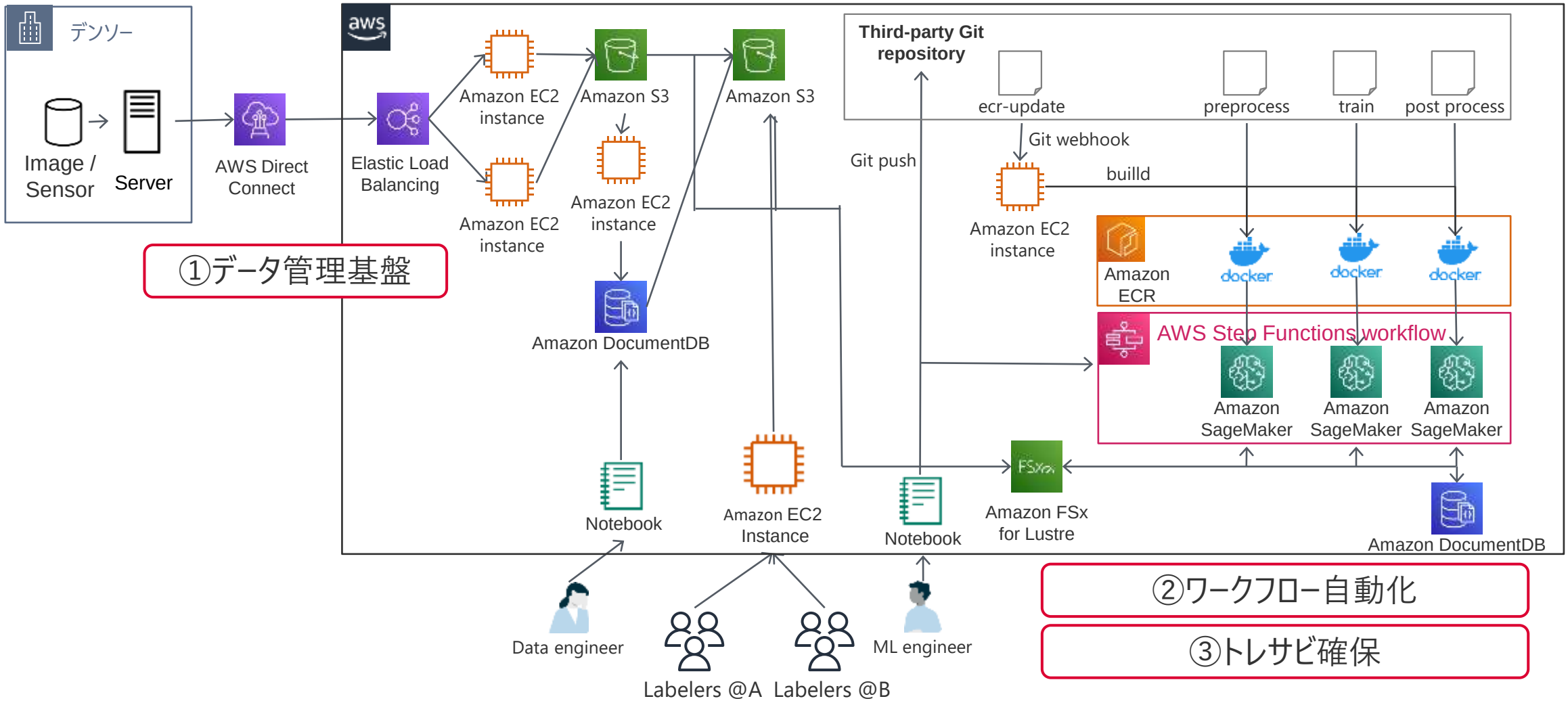
コスト算出手順
- 条件を伝える
  - 物理サーバの要件（マシンスペック、使用時間、台数、ストレージ）
  - 各作業にあたるITスタッフ工数
- 試算結果を算出
  - オンプレミス v.s. EC2 on AWS v.s. Amazon SageMaker の比較

結果
- データ管理基盤やワークフロー自動化を目指すとシステムも複雑となることもあり、運用工数やセキュリティ・コンプライアンス管理も含めた経済性評価を実施すると、AWSのマネージドサービス(Amazon SageMaker)を利用した方が費用対効果が高いことが示された

クラウドエコノミクス / Amazon SageMaker TCO Calculatorのうれしさ
- TCO (Total Cost of Ownership）での比較に加えて、目には見えにくい運用工数やセキュリティ・コンプライアンス管理工数といった視点での比較ができた

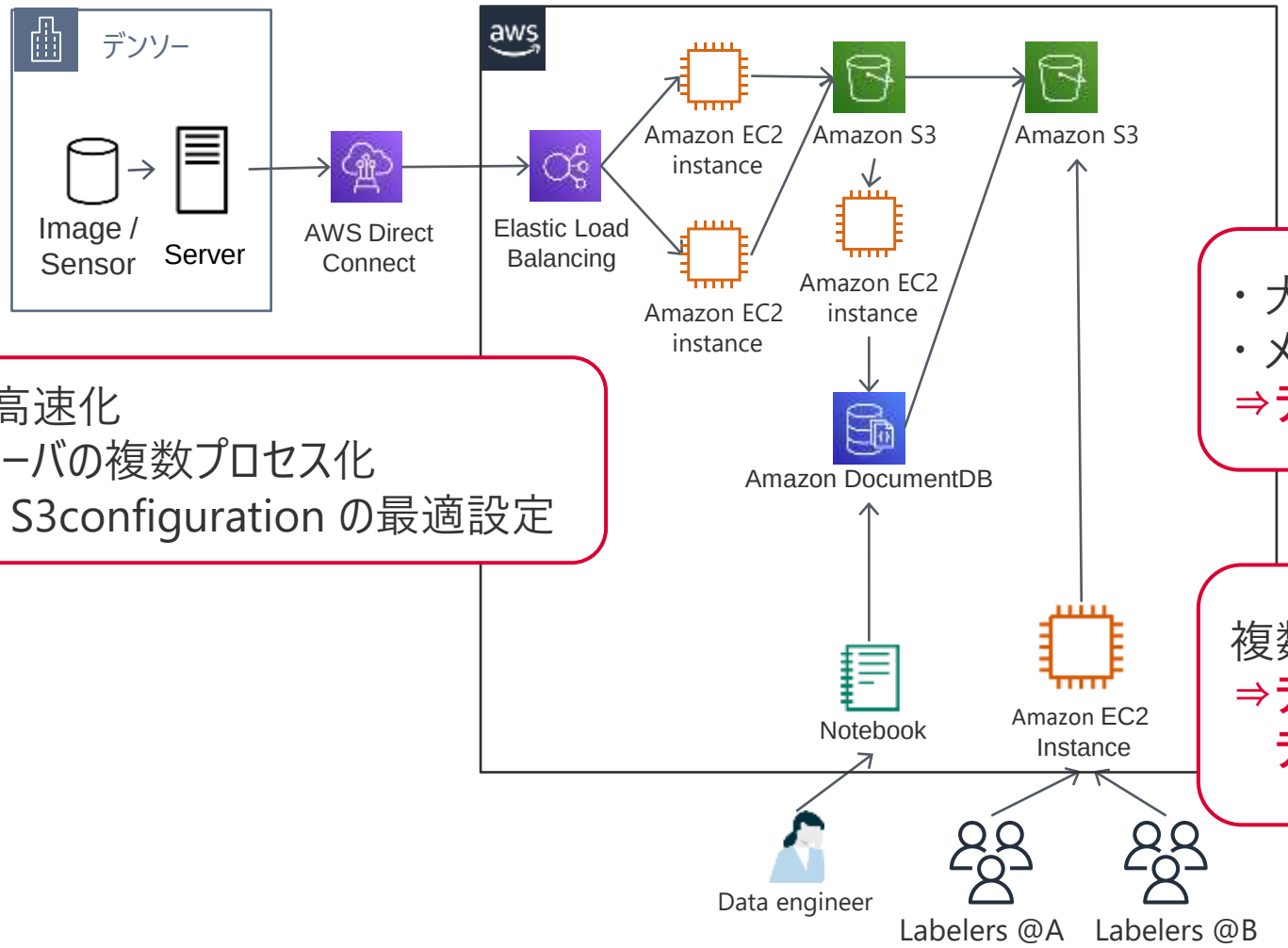
強力なHWリソースを確保した場合はコスト面でAWSに優位であることを示した

MLPL アーキテクチャとポイント



ポイント：　①データ管理基盤、②ワークフロー自動化、③トレサビ確保

データ管理基盤



アップロード高速化
・送信元サーバの複数プロセス化
・AWS CLI S3configuration の最適設定

・大きな画像はAmazon S3で管理
・メタデータをAmazon DocumentDBへ
⇒**データ/ラベルを一元管理**

複数拠点からアクセス可能
⇒**データエンジニアの**
データハンドリングコスト削減

データ管理基盤の実現により、データの一元管理、ハンドリング工数の削減を実現

ワークフロー自動化　機能のコンポーネント化（Amazon SageMakerの採用）

機能のコンポーネント化
モデル開発の前処理・学習・後処理などを
コンテナイメージとしてコンポーネント化し、Amazon SageMakerから実行可能に

副次的な大切な効果

- Amazon SageMaker スポットインスタンス＋自動再開機能
  スポットのコスト削減を享受しつつ学習を継続
  - スポットで停止したとしてもSAGEMAKER_PROGRAM
    で指定したプログラムを同一パラメタで自動再実行してくれる
  - 学習履歴があった場合に継続学習できるように
    内部コードを書いておけばよい

  ⇒**モデル学習コストを50%削減**
  スポットインスタンス起動の手間を削減

- 豊富なAWSリソースによりコンテナ分散処理も容易

⇒**1インスタンスでは3日かかる推論を3時間で実行**

Dockerfile

```
# Dockerfile for SageMaker
FROM yourbaseimage:tag
# install the SageMaker Training Toolkit
RUN pip3 install sagemaker-training
# copy the training script inside the container
COPY train.py /opt/ml/code/train.py
# define train.py as the script entry point
ENV SAGEMAKER_PROGRAM train.py
```

疑似コード

```
# SageMaker スポットインスタンスジョブ実行
from sagemaker.estimator import Estimator

hyperparameters = {"batch-size": 256, "learning-rate": 0.0001, ...}
estimator = Estimator(image_uri="your_own_container_uri",
                      role="SageMakerRole",
                      hyperparameters=hyperparameters,
                      instance_count=1,
                      instance_type="ml.p3.2xlarge"
                      use_spot_instances=True,
                      max_wait=5*24*60*60)
estimator.fit()
```

Amazon SageMakerの作法に従いコンテナイメージとしてコンポーネント化を実現

ワークフロー自動化　大量データの効率的な利用

大量データの利用方法

- Amazon S3からデータダウンロードを毎回行うには、
  データ容量が大き過ぎる
  ⇒ **Amazon FSx for Lustreを活用**

- Amazon SageMakerで
  Amazon FSx for Lustreを使用するのは容易
  - LustreIDとマウントポイントを指定
  - Inputsの引数に追加

副次的な大切な効果

- 複数ジョブでデータを共有できる場合、
  効率的に並列実行可能

疑似コード

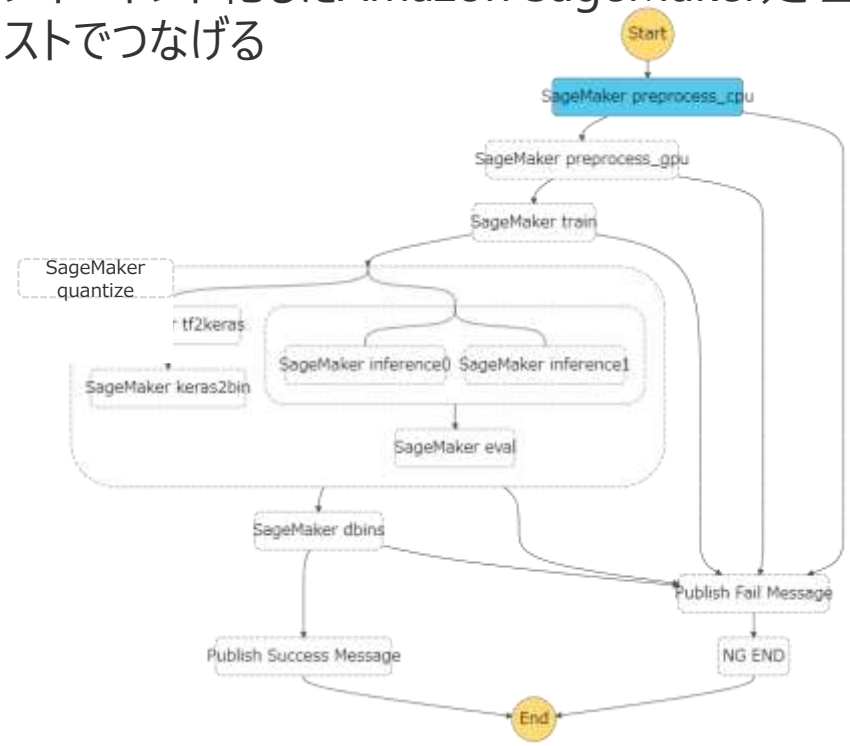
```
# Lustre のマウント
from sagemaker.inputs import FileSystemInput

fs_id = "fs-abcdefghijklmnopq"
mount_name = "abcdefgh"
s3_prefix = "root/data_folder"
fs_access_mode = 'rw'  # read and write
fs_type = 'FSxLustre'

fs_directory_path = "/{}/{}".format(mount_name, s3_prefix)
data_lustre = FileSystemInput(file_system_id=fs_id,
                              file_system_type=fs_type,
                              directory_path=fs_directory_path,
                              file_system_access_mode=fs_access_mode)
# Lustre の/root/data_folder がコンテナの/opt/ml/input/data/train にマウント
data_channels = {'train': data_lustre}
...
# ジョブ実行時に inputs の引数に追加すればよい
estimator.fit(inputs=data_channels, logs=True, wait=False)
```

Amazon Fsx for Lustreの採用により大量データを効率的に利用

ワークフロー自動化　AWS Step Functions のワークフロー記述

ワークフロー記述方法

- AWS Step Functionsから
  Amazon SageMakerを呼び出すのは容易
  - pythonで記載できる
- コンポーネント化したAmazon SageMaker処理を
  リストでつなげる



疑似コード

```
# Stepfunctions による SageMaker 呼び出し
import stepfunctions
from stepfunctions import steps
from stepfunctions.workflow import Workflow
import sagemaker

# コンポーネント化した処理を定義（定義済み）
preprocess = sagemaker.Estimator(image_uri, ...)
preprocess_step = steps.TrainingStep(estimator=preprocess,
                                     data=data_path,
                                     wait_for_completion=True, ...)
train = sagemaker.Estimator(image_uri, ...)
train_step = steps.TrainingStep(estimator=train,
                                data=data_path,
                                wait_for_completion=True, ...)

# フロー構築・実行
chain_list = [preprocess_step, train_step]
workflow_definition = steps.Chain(chain_list)
workflow = Workflow(definition=workflow_definition, ...)
workflow.execute()
```

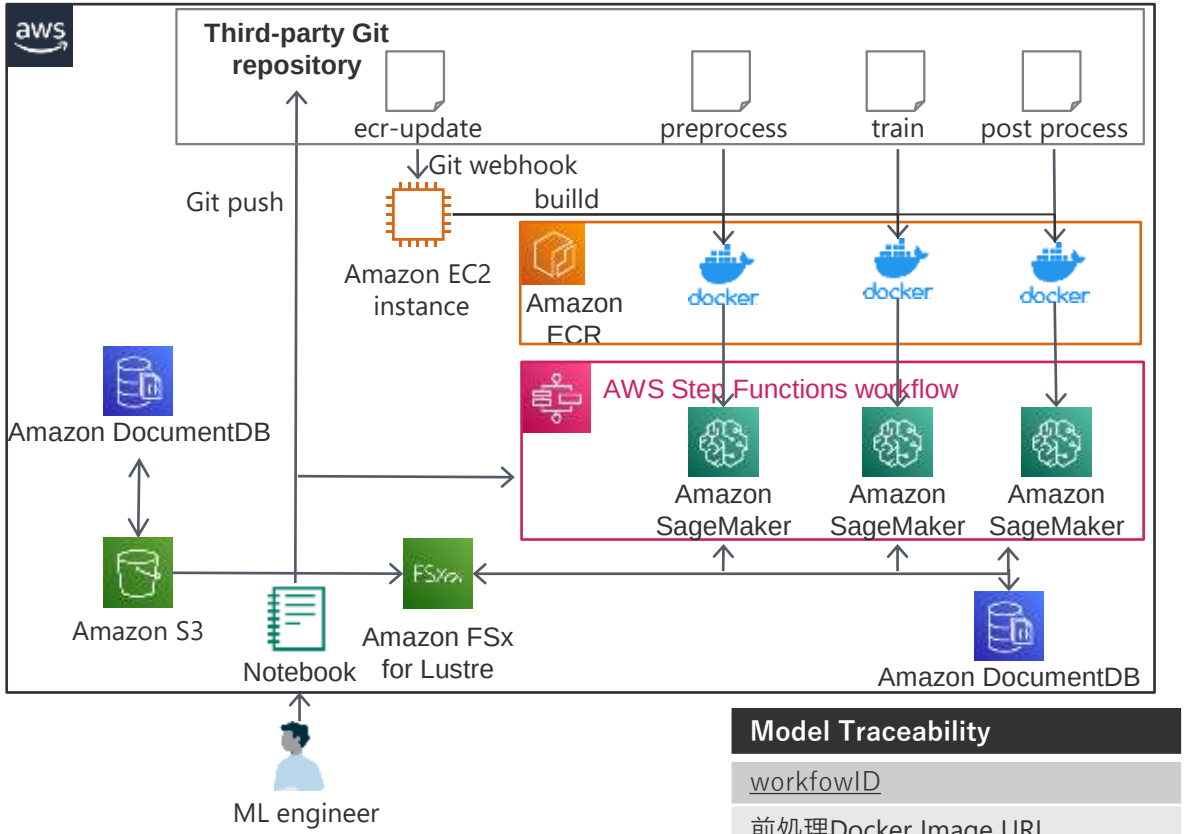
AWS Step Functionsでワークフローとして実行可能に

トレサビ確保　データ/ラベル と ワークフローの管理方法

データ / ラベル
- Amazon DocumentDBで管理

ワークフロー
- ワークフローごとに一意のworkflowIDを付与し
  Amazon DocumentDBに記録
  - workflowIDからデータの入出力などが確認可能

ポイント
- Latestタグのコンテナイメージを管理していたとしても、
  過去のバージョンの解析・分析が難しい
  ⇒ワークフロー実行（モデルリリース）の度に
  immutableタグを切ることで管理する
- Bitbucketでタグを切ると、
  CI/CDツールが同タグでビルドしECRにプッシュ
  - フローでは指定タグのイメージを仕様



Model Traceability
workfowID
前処理Docker Image URI
学習データセット Amazon S3 Path
評価データセット Amazon S3 Path
学習Docker Image URI
学習結果モデル Amazon S3 Path
…

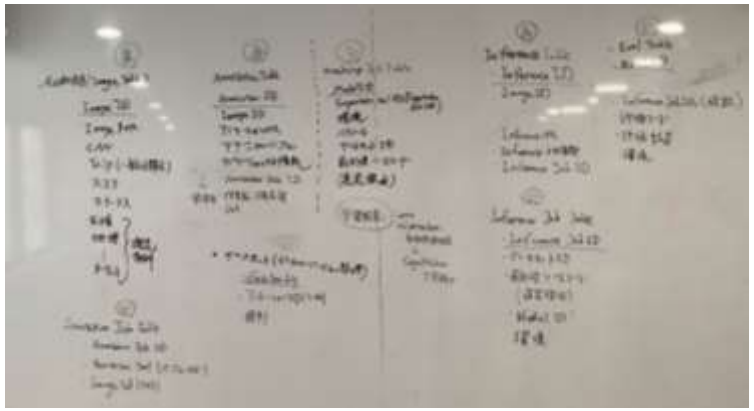
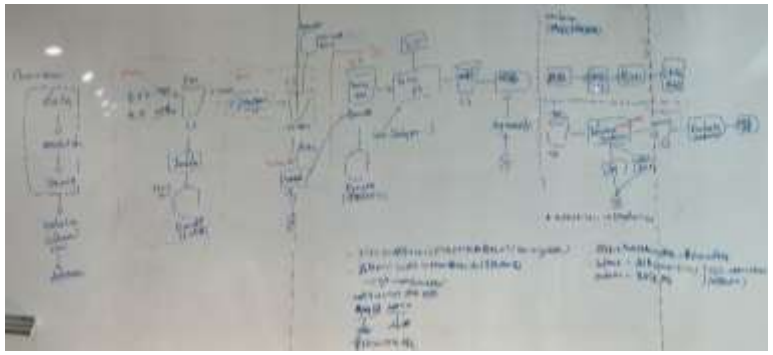
ワークフロー実行に追加処理としてトレサビ情報も自動取得
ワークフロー実行（モデルリリース）の度にコードタグを切ることで実行環境とコードを保存

全体の効果

MLPLのポイント	目標	効果
強力なHWリソース	オンプレミス/AWSの対比コストが安くなること	• オンプレミス/AWSの比較を行いAWSがコストメリットがあることを示した
データ管理基盤	現行プロセス比 60%以下にする	• 現行プロセス比 55%に低減 • データのハンドリングを確保することで削減目標を達成
ワークフロー自動化	現行プロセス比 70%以下にする	• 現行プロセス比 66%に低減 • Amazon SageMaker / AWS Step Functionsによるワークフロー記述により削減目標を達成
トレサビ確保	トレサビ確保を可能とする	• トレサビ情報が確保可能な環境の構築できた • データ/ラベル管理、ワークフローの一連の情報をAmazon DocumentDBに保存

苦労した点

- 全体像/課題の共有
  - プロセスが膨大で、Data / Model Line それぞれの現状、あるべき姿を議論するのに時間を要した
    - AWSソリューションアーキテクトも含めて、White Boardingで丸3日議論を重ねた
      （結果としては非常によかった）

- MLPL の普及活動
  - 作ったものを便利だと理解してもらうまでが難しい。理解してもらってからは好評。

5

まとめと今後

まとめと今後

まとめ

- 高度運転支援向けの画像認識の開発の特徴である多量なデータを用いて多様なモデルを開発するため、AIの透明性獲得のためトレサビを確保すること、生産性を向上すること、実現することを目的としたMLPL(Machine Learning Production Line) をAWS上で構築した

- MLPLの開発のポイントをトレサビ確保、生産性の観点から、「強力なHWリソース」、「データ管理基盤」、「ワークフロー自動化」、「トレサビ確保」をあげて、AWSでの実現方法を示した

今後

- Amazon SageMaker Ground Truth を用いたアノテーションの効率化の検討
- MLPL上で次世代の開発向けの追加機能の折込み

Thank you!

新原　竜馬

高度運転支援向け画像認識のための
Machine Learning Production Line