AWS
Black Belt
Online Seminar

【AWS Black Belt Online Seminar】
Amazon SageMaker

Makoto Shimura, Analytics Specialist Solution Architect
Amazon Web Services Japan, K. K.

2018.03.08

自己紹介

志村 誠

- アナリティクススペシャリスト ソリューションアーキテクト
  - データ分析・機械学習系サービスを担当
  - 前職はログ解析基盤構築・データ分析等
  - 好きなサービス
    - Amazon Athena
    - AWS Glue
    - そして Amazon SageMaker

内容についての注意点

- 本資料では2018年03月08日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

Agenda

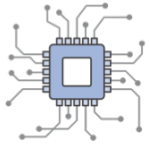
- 機械学習システムでよくある問題
- SageMaker の概要
- SageMaker の使いかた
- 実践的な活用法
- その他

機械学習システムでよくある問題

機械学習の流れ

学習

大量（～数十コア）の GPU
大規模データの処理
試行錯誤の繰り返し



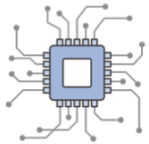
推論

大量（～数百台）の GPU または CPUインスタンス
継続的なデプロイ
さまざまなデバイスで動作

機械学習の流れ

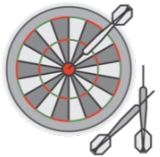
開発 & 学習

学習に使うコードを記述　大量の GPU
小規模データで動作確認　大規模データの処理
試行錯誤の繰り返し



推論

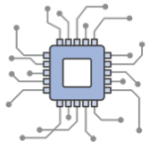
大量の GPU と CPU
継続的なデプロイ
さまざまなデバイス
で動作



機械学習システムの構成

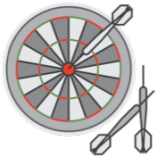
開発 & 学習

データサイエンティストが**開発環境**で作業
開発と学習を同じ 1 台のインスタンスで実施
Deep Learning であれば GPU インスタンスを使用



推論

エンジニアが**プロダクション環境**に構築
エンドポイントを作成
通常の API サーバ
A/B テストの仕組み



機械学習の流れ

開発

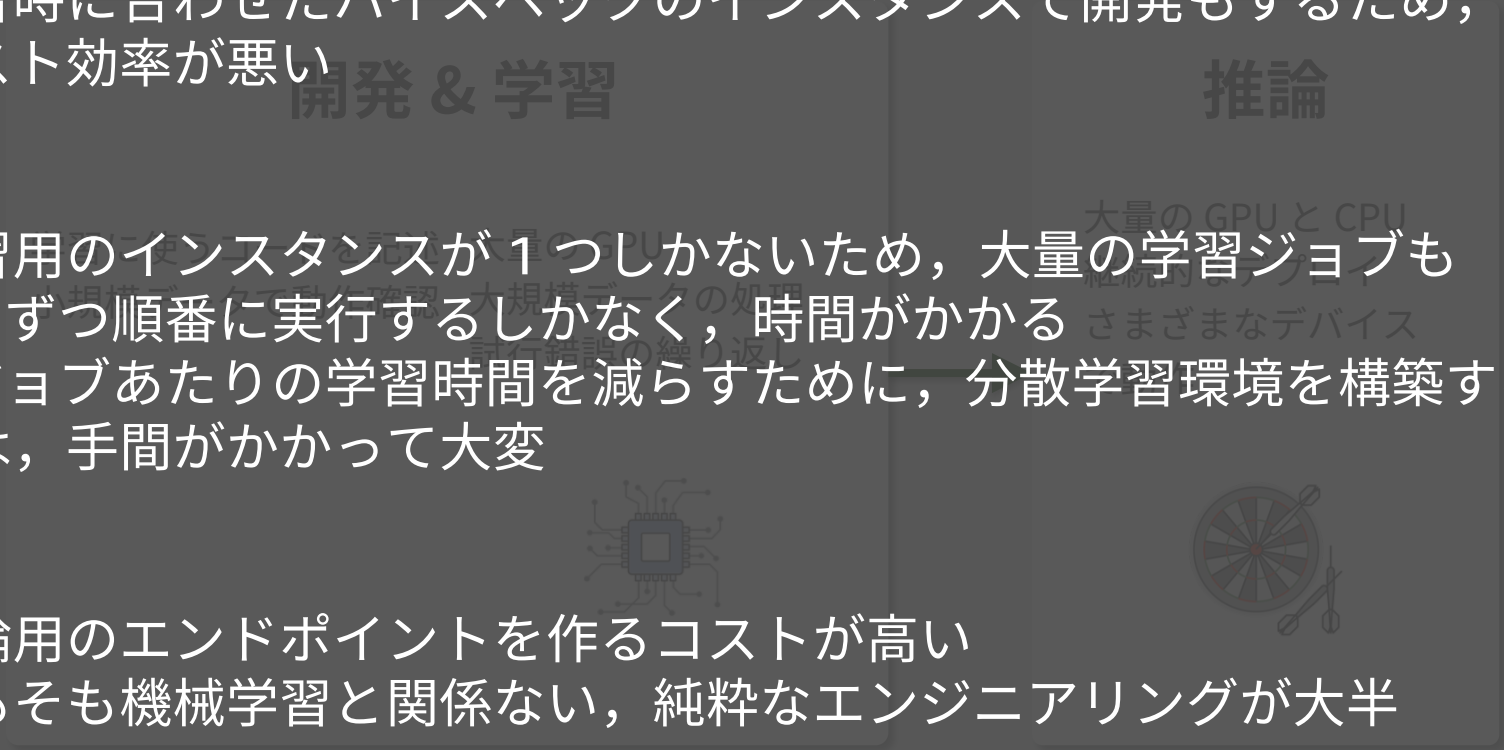
- 学習時に合わせたハイスペックのインスタンスで開発もするため，コスト効率が悪い

学習

- 学習用のインスタンスが１つしかないため，大量の学習ジョブも１つずつ順番に実行するしかなく，時間がかかる
- １ジョブあたりの学習時間を減らすために，分散学習環境を構築するのは，手間がかかって大変

推論

- 推論用のエンドポイントを作るコストが高い
- そもそも機械学習と関係ない，純粋なエンジニアリングが大半



Amazon SageMaker

機械学習モデルを大規模に構築、トレーニング、デプロイ

SageMaker の概要

SageMaker とは

- 機械学習システムでよくある問題を解消し，データサイエンティストやエンジニアが素早くプロセスを回せるようにするためのサービス
- バージニア北部，オハイオ，オレゴン，アイルランドの 4 リージョンで提供

ノートブックインスタンス

ノートブックの AWS データを参照し、トレーニングジョブを通じてアルゴリズムを駆使したモデルの作成を行います。

ジョブ

トレーニングジョブをデスクまたはリモートで追跡します。高パフォーマンスの AWS アルゴリズムを活用します。

モデル

ジョブ出力からホストするモデルを作成するか、外部でトレーニングされたモデルを Amazon SageMaker にインポートします。

エンドポイント

開発者が本番稼働用環境で使用するエンドポイントをデプロイします。エンドポイントを介した A/B テストモデルバリアント。

SageMaker が提供するアーキテクチャ

開発

Jupyter Notebook

- コンソールから簡単にノートブックインスタンスを起動
- 開発用マシンは最低限のスペック
- 主要ライブラリは**プリインストール済**で，後から追加も可能

学習

Docker コンテナ

- SageMaker の API を叩いて，学習ジョブを実行
- 開発環境と学習環境を**完全に分離**
- 複数の学習ジョブを**同時実行**可能
- **分散学習**ジョブも簡単に実行可能

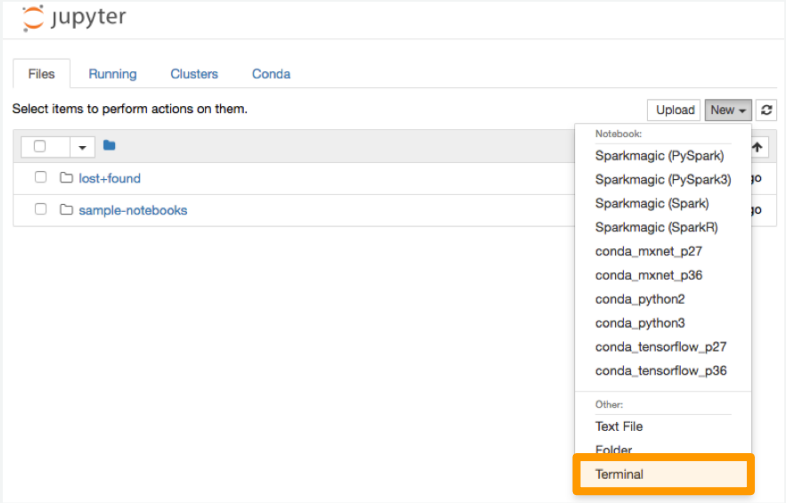
推論

Docker コンテナ

- SageMaker の API を**叩くだけ**で，エンドポイントが作成
- **オートスケーリング**機能や，**A/Bテスト**機能を提供

開発

- マネジメントコンソールからインスタンスを作成し，リンクをクリックして直接 Jupyter Notebook を起動
  - ターミナルを起動可能
  - 外向きのインターネットアクセスがあり，後から開発に必要なパッケージをインストール可能
  - インスタンスタイプは 3 種類のみ*
    - ml.t2.medium
    - ml.m4.xlarge
    - ml.p2.xlarge

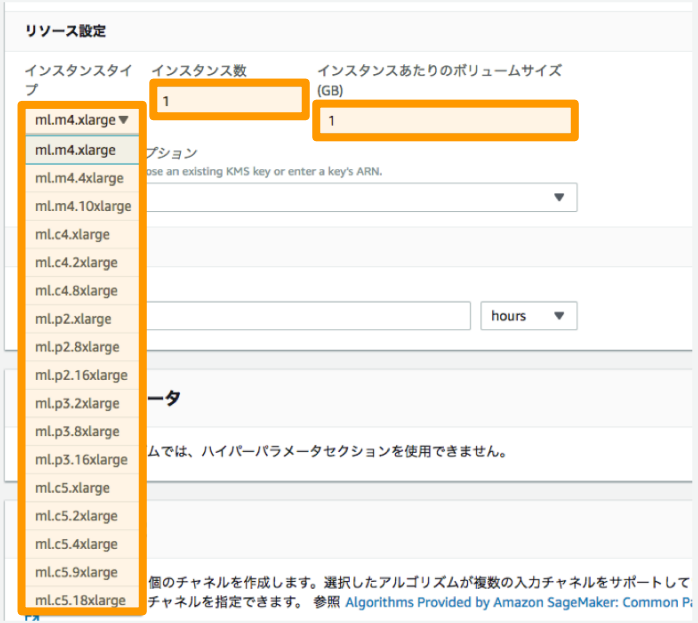


* あくまで開発環境として使うため．P2 インスタンスは GPU を使ったコードの挙動確認のためにのみ利用

学習

- **CreateTrainingJob** API を叩くと，Docker コンテナが立ち上がって学習ジョブを実行し，終わると自動でコンテナが終了

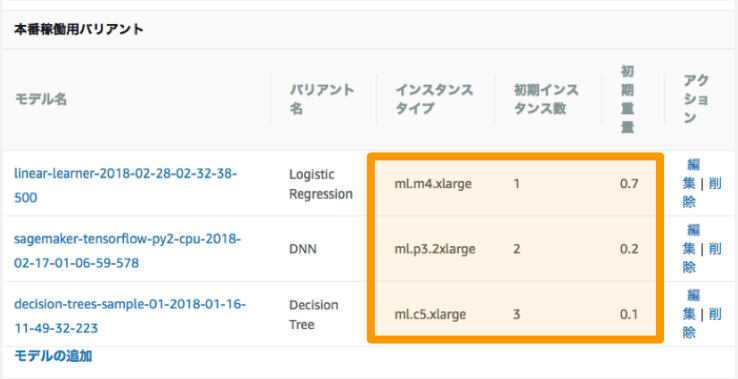
  - コンテナを動かすインスタンスタイプ，インスタンス数，各インスタンスのボリュームサイズを指定可能
  - インスタンス数を 2 以上にすれば，自動で分散学習を実行
  - インスタンスタイプは m4, c4, c5, p2, p3 から選択可能
  - 入力データの読み込み元と，学習済モデルの出力先は，いずれも S3



https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html

推論

- **`CreateEndpoint`** API を叩くと，Docker コンテナが立ち上がってモデルを読み込み，エンドポイントができる
  - 読み込むモデルを複数指定して，AB テストを実施可能
  - 同じエンドポイントに対して，新しい設定を適用することで，エンドポイントの更新が可能
  - ターゲットメトリクスを指定して，エンドポイントのオートスケールが可能



https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateEndpointConfig.html
https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateEndpoint.html
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/API_UpdateEndpoint.html
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/API_UpdateEndpointWeightsAndCapacities.html

aws

SageMaker の使いかた

SageMaker を操作するときの 2 つの方法

AWS SDK

- **基盤エンジニア**向け
- プロダクション環境で，定期学習ジョブの実行や，エンドポイントへのモデルのデプロイ等を実施するためのもの

SageMaker SDK

- **データサイエンティスト**向け
- 通常の AWS SDK とは別の，SageMaker だけのための SDK で，AWS SDK を scikit-learn ライクにラップしたインターフェース
- Python 版とSpark 版がある
- Jupyter Notebook 上で，機械学習モデルの高速な開発と学習を行うためのもの

https://github.com/aws/sagemaker-python-sdk
https://github.com/aws/sagemaker-spark

それぞれの方法の利用イメージ

AWS SDK

- `create-endpoint`
- `create-notebook-instance`
- `create-training-job`
- `delete-endpoint`
- `delete-notebook-instance`
- `describe-endpoint`
- `describe-notebook-instance`

…

SageMaker SDK

```
estimator = TensorFlow(…)
estimator.set_hyperparameters(…)
estimator.fit(…)
predictor = estimator.deploy(…)
Predictor.predict(…)
```

SageMaker を使った機械学習モデルの開発

1. SageMaker のビルトインアルゴリズムを使う

2. Tensorflow/MXNet/Spark を使う

3. それ以外のやり方で開発を行う

SageMaker を使った機械学習モデルの開発

1. SageMaker のビルトインアルゴリズムを使う
   → **学習用データが必要**

2. Tensorflow/MXNet/Spark を使う
   → **学習用データ**と，**学習用コード**が必要

3. それ以外のやり方で開発を行う
   → **学習用データ**，**学習用コード入りのコンテナ**が必要

SageMaker の
ビルトインアルゴリズムを使う

現在サポートされているアルゴリズム一覧

- Linear Learner
- Factorization Machines
- XGBoost
- Image Classification
- seq2seq
- K-means
- PCA
- LDA
- Neural Topic Model
- DeepAR Forecasting *New!*
- BlazingText (word2vec) *New!*

https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html

aws

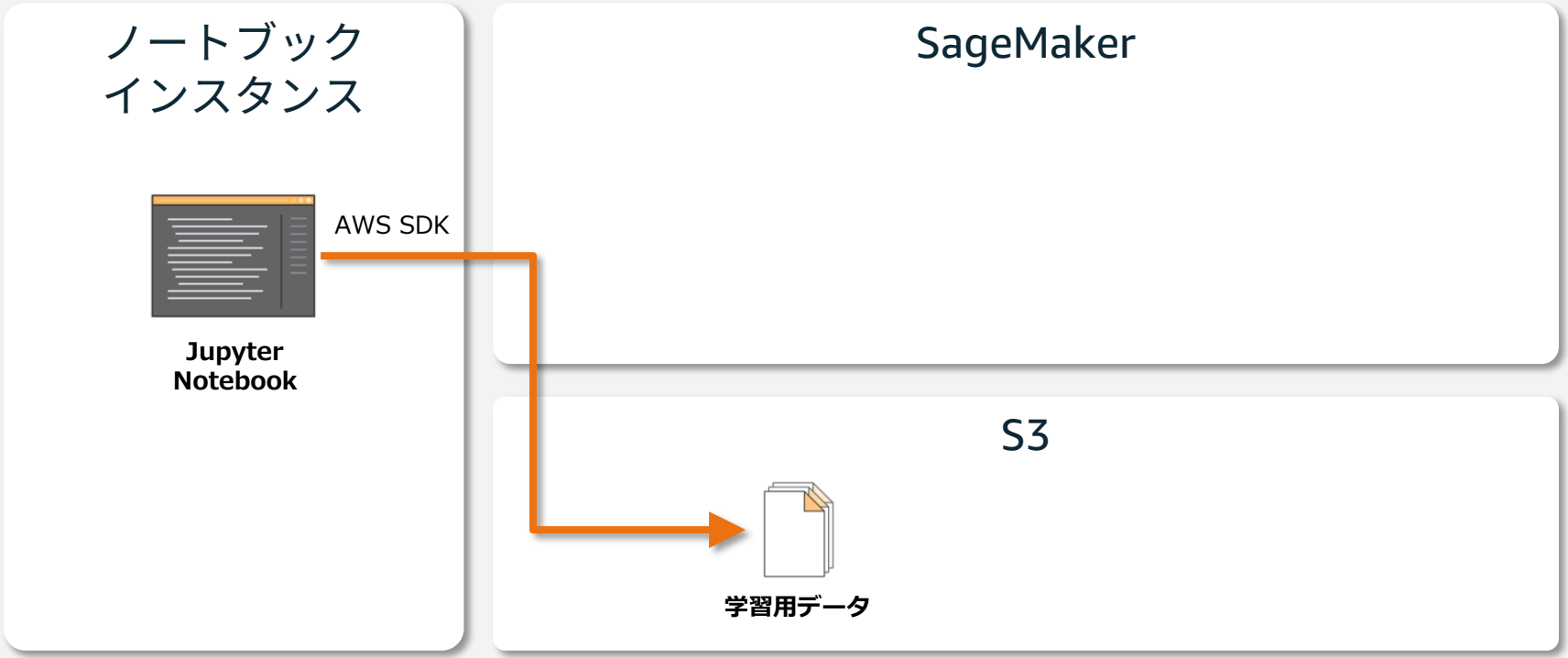
Image Classification

- アルゴリズムとしては，ResNet を使用
  - ResNet は，非常に精度の良い CNN ベースの著名なネットワーク
  - ILSVRC 2015 という画像認識コンテストで 1 位を獲得
- 転移学習が可能
  - 画像認識では定番の，ImageNet データで学習済みのパラメタを内部に保持しており，これを用いて転移学習を実施することが可能
  - ハイパーパラメータの設定で，**`use_pretrained_model`** パラメタを 1 にすればよい（0 にした場合は，全パラメタにランダムな値が割り振られる）

https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/imageclassification_caltech/Image-classification-transfer-learning.ipynb
https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/image-classification.html
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/IC-Hyperparameter.html

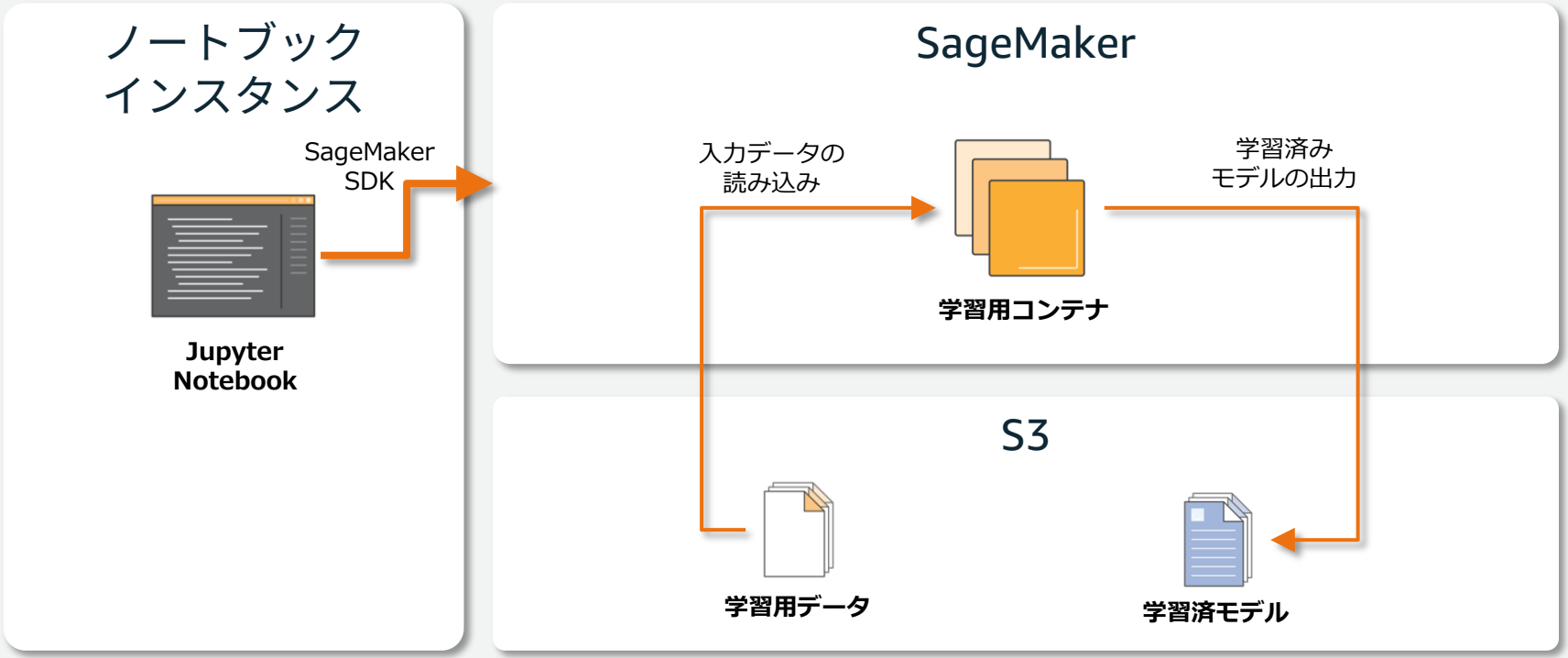
開発

ノートブックから，**学習用データ**を作成して，S3 に置く



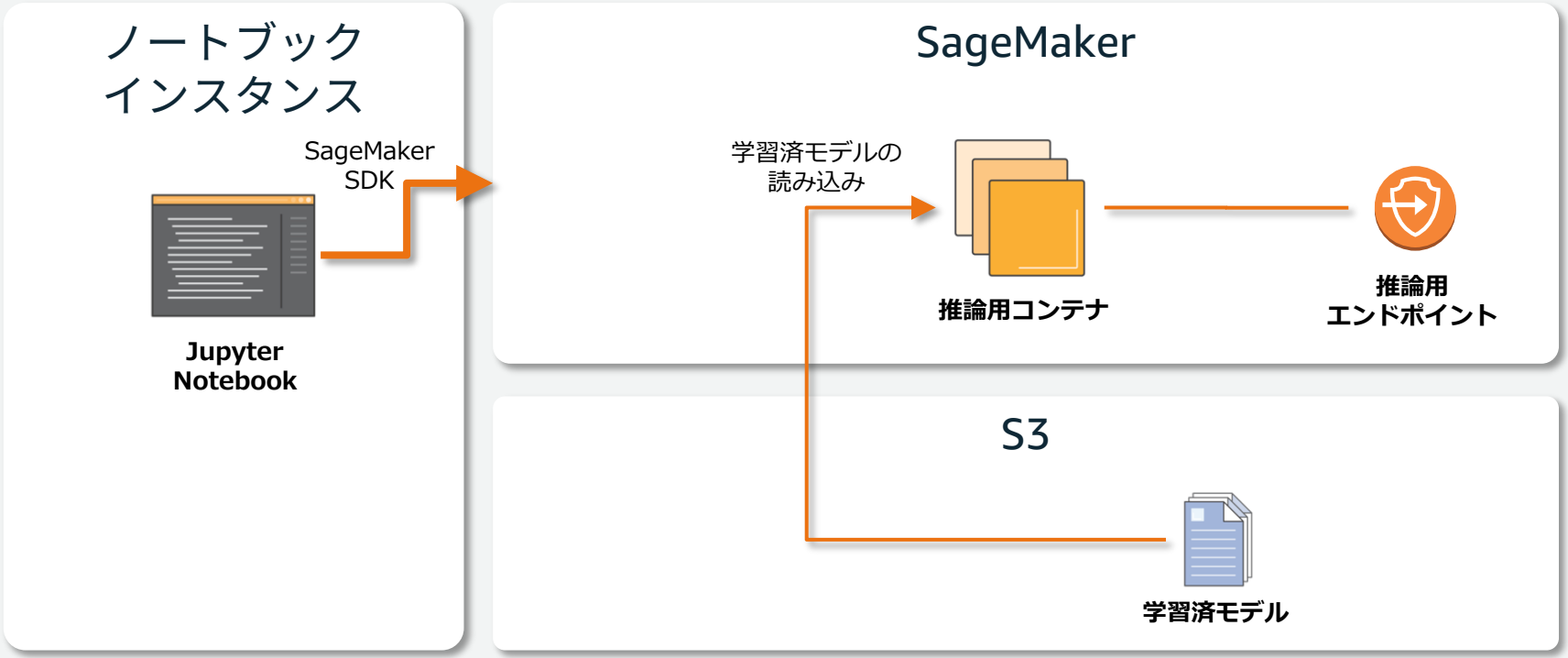
学習

入力データ，モデル出力場所を指定して，`estimator.fit()` を実行.
AWS が提供するコンテナが立ち上がり，学習ジョブを実行



推論

学習済モデルを指定して，`estimator.deploy()` を実行
AWS が提供するコンテナが立ち上がり，推論エンドポイントを作成



```
from sagemaker.estimator import Estimator

estimator = Estimator(container_id,
                      role='SageMakerRole',
                      train_instance_count=1,
                      train_instance_type='ml.c4.xlarge',
                      output_path='s3://path/to/output/data/',
                      sagemaker_session=session)

estimator.fit({'train': 's3://mpath/to/train/data/'})

estimator.deploy(initial_instance_count=1,
                 instance_type='ml.m4.xlarge')
```

https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html

aws

Tensorflow/MXNet/Spark を使う

Tensorflow を使うときに必要なインタフェース

学習用コード

- **`model_fn`**: モデル，損失関数定義，最適化関数等を記述
  - **`estimator_fn`**: 既存の **`tensorflow.estimator`** を使う場合はこちら
  - **`keras_model_fn`**: 既存の **`tf.keras`** を使う場合はこちら
- **`train_input_fn`**: 学習データロードと前処理を記述
- **`eval_input_fn`**: 評価データロードと前処理を記述
- **`serving_input_fn`**: 学習済モデルの保存処理を記述

推論用コード

- **`input_fn`**: 入力データに対する前処理を記述
- **`output_fn`**: 予測結果に対する後処理を記述

https://docs.aws.amazon.com/sagemaker/latest/dg/tf-training-inference-code-template.html

MXNet を使うときに必要なインタフェース

学習用コード

- **`train`**: モデル，損失関数定義，最適化関数等を記述
- **`save`**:学習済モデルの保存処理を記述

推論用コード

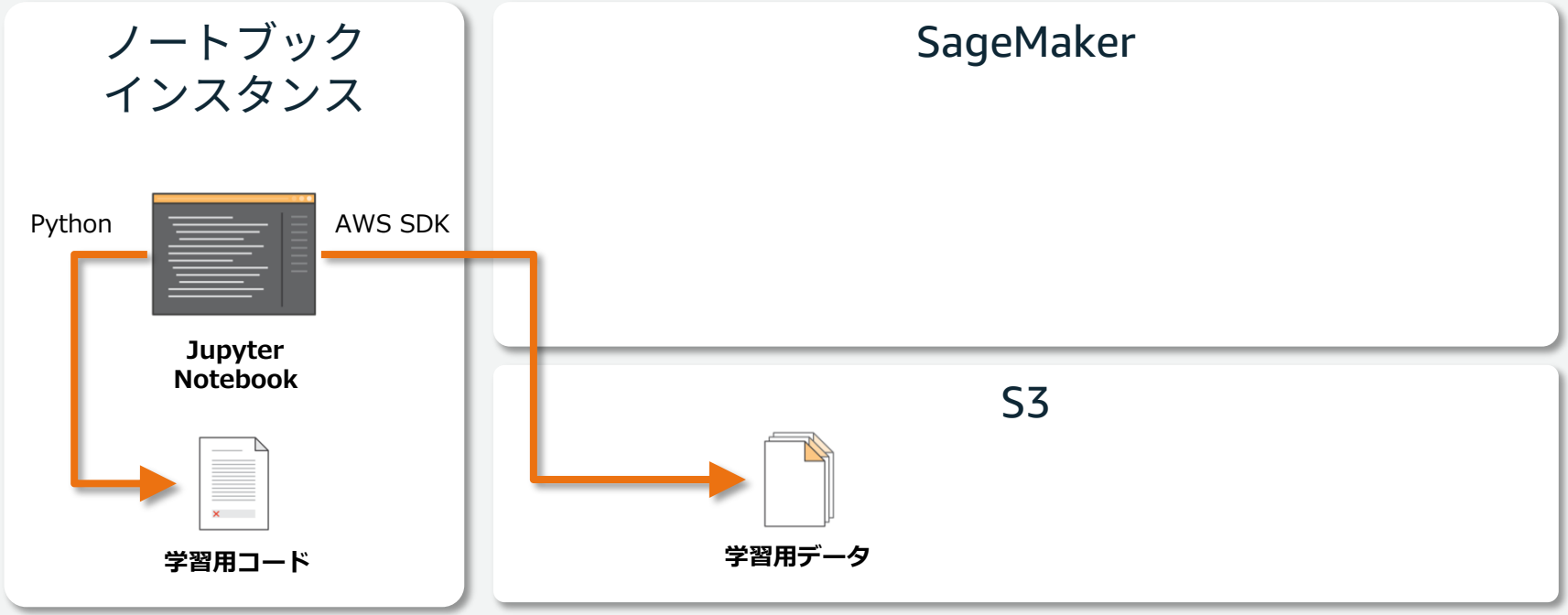
- **`model_fn`**: 学習済みモデルのロード処理を記述
- **`transform_fn`**: リクエストデータの予測処理を記述
- そのほか，Gluon 用インタフェース，Module モデル用インタフェースも定義することが可能

https://docs.aws.amazon.com/sagemaker/latest/dg/mxnet-training-inference-code-template.html

aws

開発

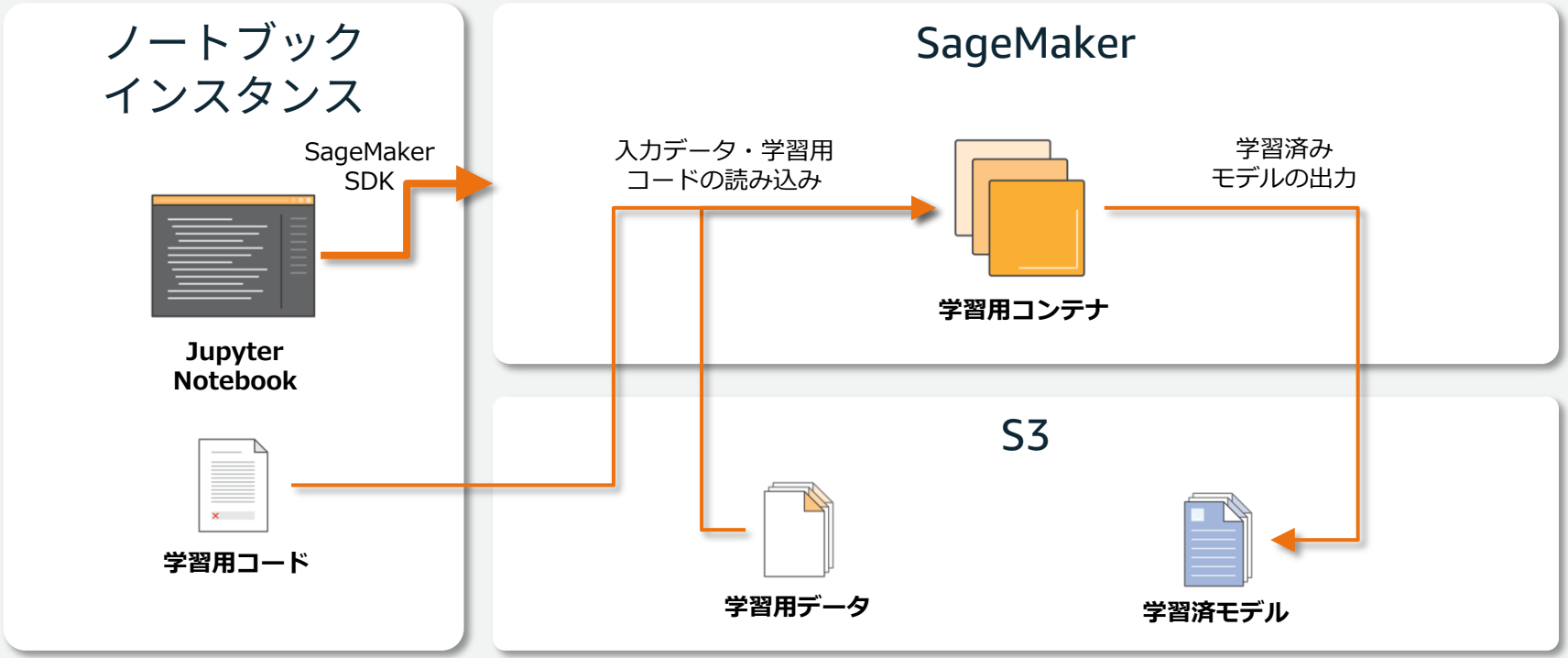
ノートブックで，**学習用コード**を作成してローカルに置き*，
学習用データを作成してS3 に置く



* 内部的は，学習を実行する際に S3 にアップロードされる．最初から S3 に tar で固めて配置してくことも可能

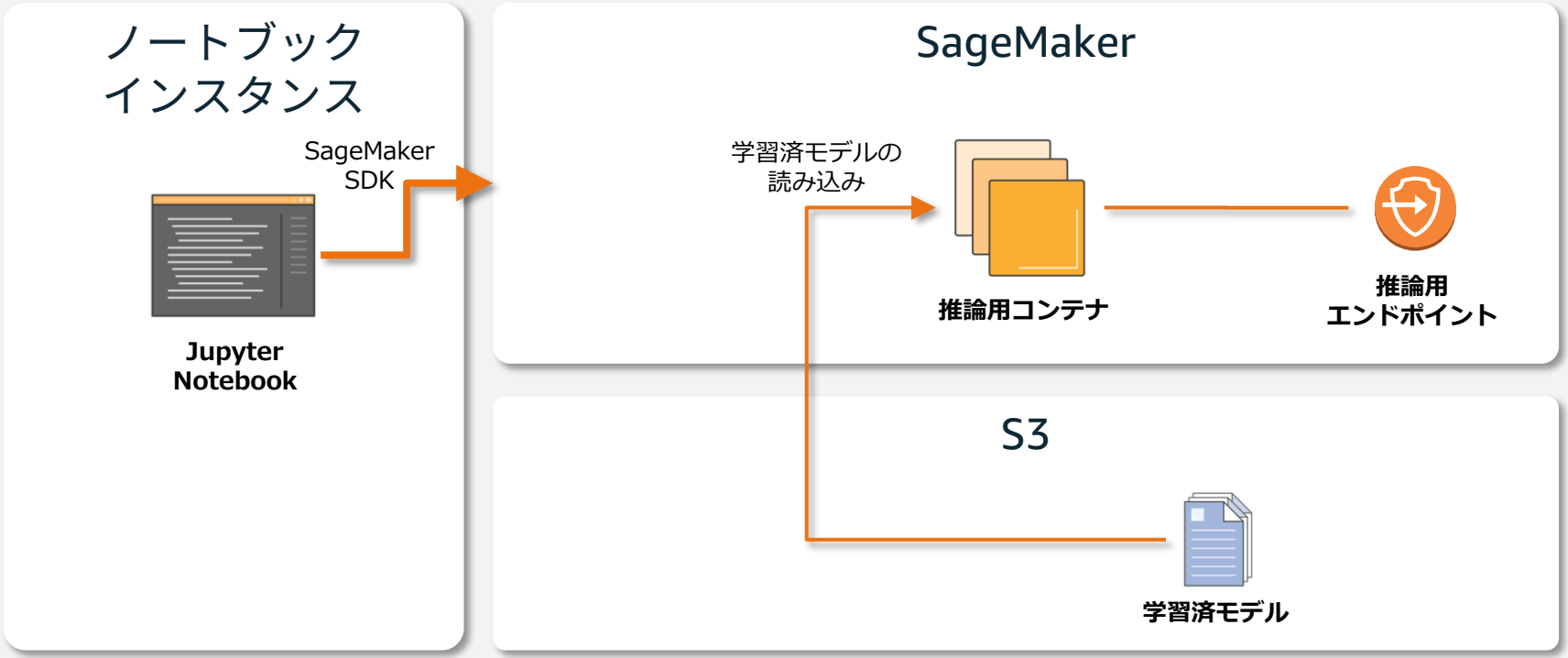
学習

入力データ，学習用コードを指定して `estimator.fit()` を実行.
AWS が提供するコンテナが立ち上がって，学習ジョブを実行



推論

学習済モデルを指定して，`estimator.deploy()` を実行
AWS が提供するコンテナが立ち上がり，推論エンドポイントを作成



```
from sagemaker.tensorflow import TensorFlow

estimator = TensorFlow(entry_point='mnist.py',
                       source_dir='/path/to/source/code/directory/'
                       role=role,
                       framework_version='1.5',
                       train_instance_count=3,
                       train_instance_type='ml.p3.16xlarge',
                       hyperparameters={'learning_rate': 0.1},
                       output_path='s3://path/to/output/data/')

estimator.fit({'train': 's3://path/to/train/data/'})

estimator.deploy(initial_instance_count=5,
                 instance_type='ml.m4.xlarge')
```

```
{
    "InputDataConfig": [
        {
            "ChannelName": "training",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3DataDistributionType": "FullyReplicated",
                    "S3Uri": "s3://sagemaker-sample-data-us-west-2/tensorflow/iris"
                }
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemakermv/artifacts"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    },
    "TrainingJobName": "sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686",
    "AlgorithmSpecification": {
        "TrainingInputMode": "File",
        "TrainingImage": "142577830533.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-py2-cpu:1.0.5"
    },
    "HyperParameters": {
        "sagemaker_program": "\"mnist.py\"",
        "checkpoint_path": "\"s3://sagemakermv/artifacts/checkpoints\"",
        "sagemaker_job_name": "\"sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686\"",
        "sagemaker_submit_directory": "\"s3://sagemakermv/customcode/tensorflow_iris/sourcedir.tar.gz\"",
        "sagemaker_region": "\"us-west-2\"",
        "training_steps": "100",
        "sagemaker_container_log_level": "20"
    },
    "ResourceConfig": {
        "VolumeSizeInGB": 30,
        "InstanceCount": 1,
        "InstanceType": "ml.c4.xlarge"
    },
    "RoleArn": "arn:aws:iam::account-id:role/SageMakerRole"
}
```

CreateTrainingJob API の
各種オプションは
Hyperparameters で指定

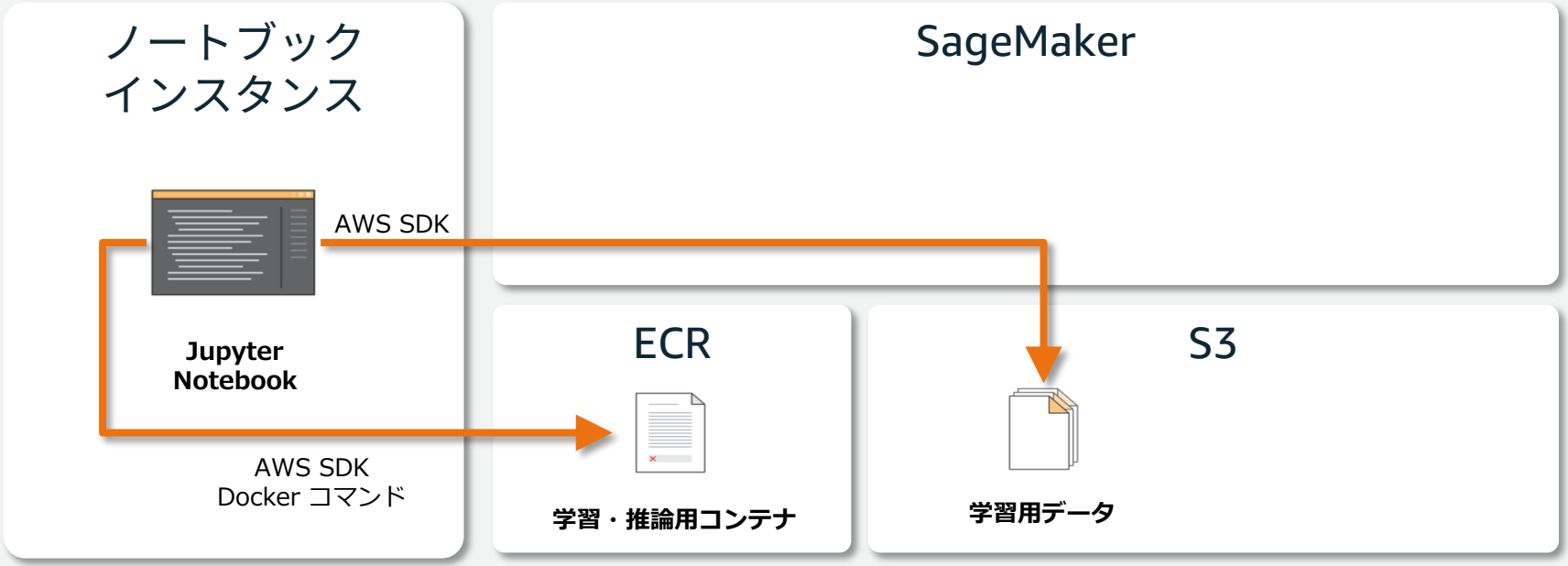
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/API_CreateTrainingJob.html
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/tf-example1-train.html

aws

それ以外のやり方で開発を行う

開発

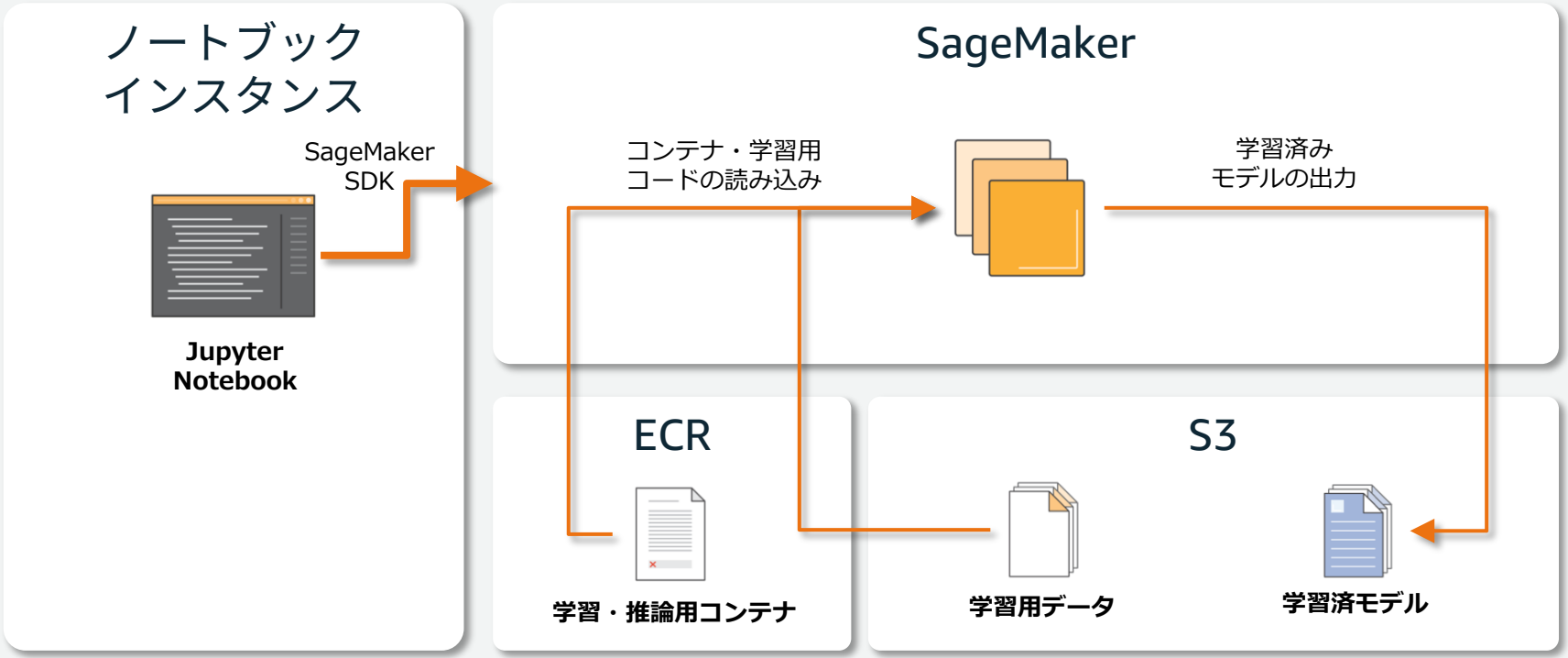
ノートブックで，**学習コードと推論用 Web サーバが入ったコンテナ**を作成して ECR* に `push` し，**学習用データ**を作成してS3 に置く



* Elastic Container Registry

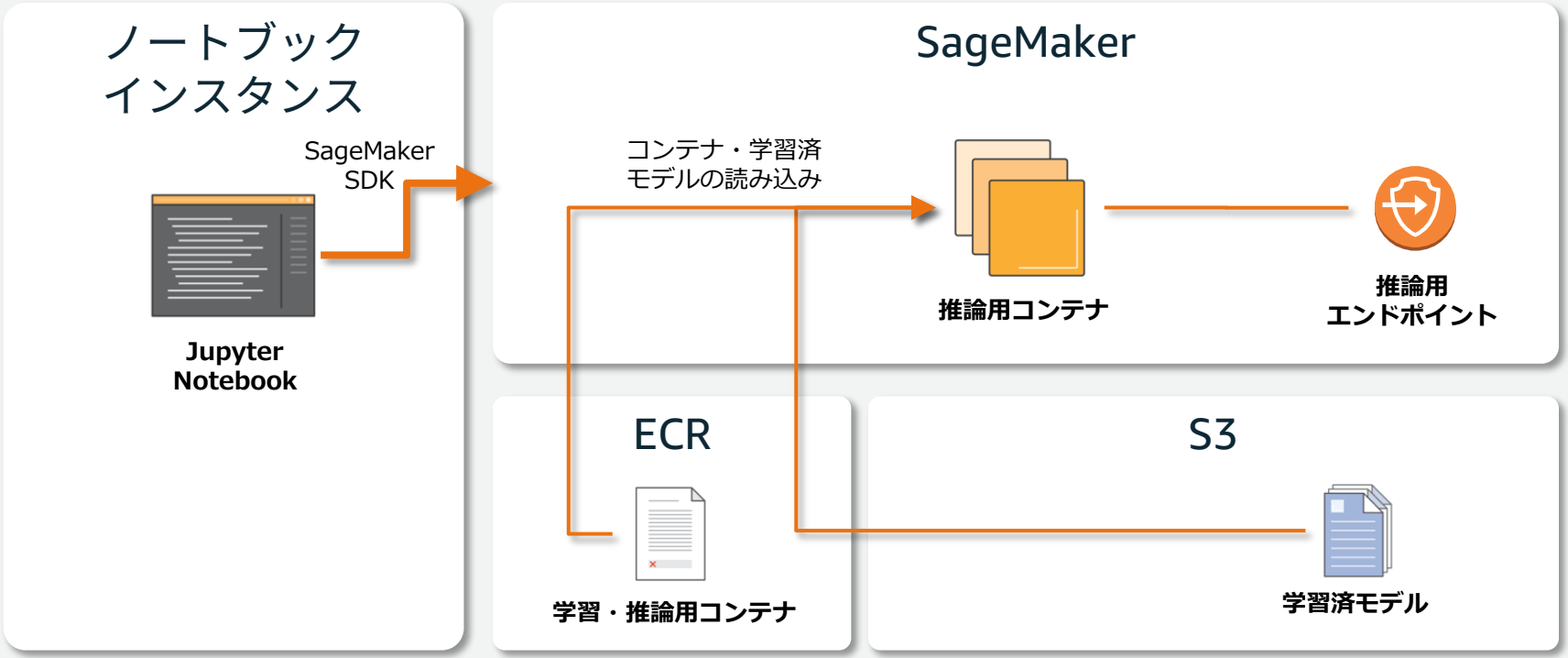
学習

入力データ，学習用コンテナ，モデル出力場所を指定して
`estimator.fit()` を実行．指定したコンテナで学習ジョブを実行



推論

学習済モデルと推論用コンテナを指定して，`estimator.deploy()` を実行．指定したコンテナを使ってエンドポイントを作成



```
from sagemaker.estimator import Estimator

estimator = Estimator(container_id,
                      role='SageMakerRole',
                      train_instance_count=1,
                      train_instance_type='ml.c4.xlarge',
                      output_path='s3://path/to/output/data/',
                      sagemaker_session=session)

estimator.fit({'train': 's3://mpath/to/train/data/'})

estimator.deploy(initial_instance_count=1,
                 instance_type='ml.m4.xlarge')
```

自前のコンテナイメージに必要なインターフェース

学習用コード: `docker run IMAGE_ID train`

- docker コンテナ内に ENTRYPOINT を指定して，その場所に **`train`** という名前のスクリプトを作成し，学習ジョブをその中に記述
- **`estimator.fit()`** を実行すると，上記の **`docker run`** コマンド経由で **`train`** スクリプトが叩かれる

推論用コード: `docker run IMAGE_ID serve`

- docker コンテナ内に ENTRYPOINT を指定して，その場所に **`serve`** という名前のスクリプトを作成し，推論用 Webサーバを記述
- **`estimator.fit()`** を実行すると，上記の **`serve`** コマンドが叩かれる
- **`predictor.predict()`** を実行すると，Web サーバ内の **`/invocations`** が叩かれる
- 学習・推論をひとつのコンテナにまとめることも可能

https://docs.aws.amazon.com/sagemaker/latest/dg/your-algorithms-training-algo.html
https://docs.aws.amazon.com/sagemaker/latest/dg/your-algorithms-inference-code.html

aws

実践的な活用法

SageMaker の開発・デプロイフロー

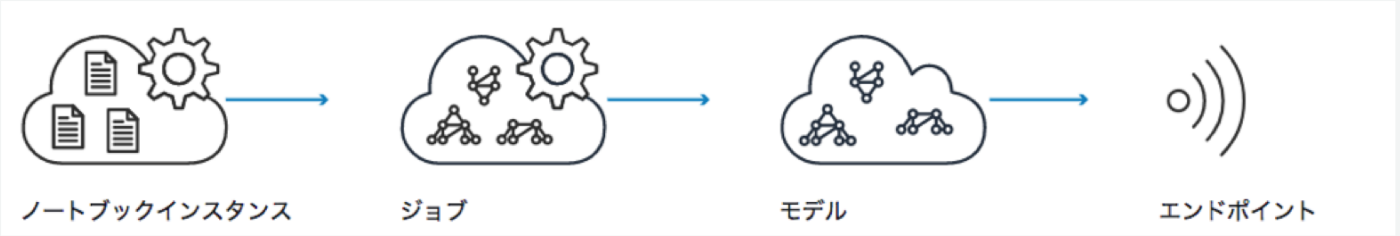
SageMaker の 3 要素は，それぞれ個別で利用可能

- 例 1: プロダクション環境がオンプレミスにすでにある場合
  - スケーラブルな学習環境としてのみ SageMaker を利用可能

- 例 2: オンプレミスに豊富な GPU クラスタを持っている場合
  - オンプレミスで学習済したモデルを AWS 上のプロダクション環境にデプロイ

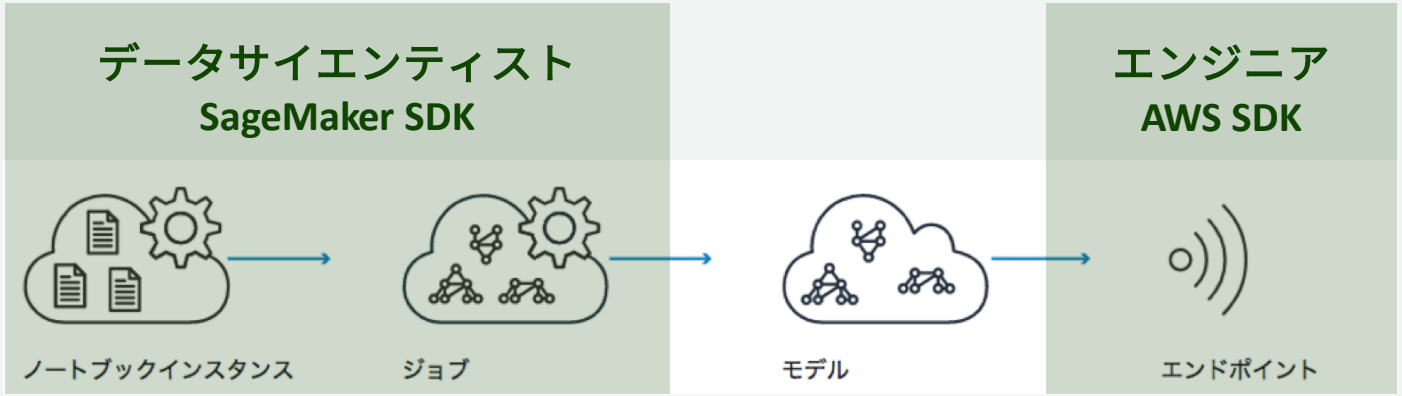
AWS SDK と SageMaker SDK の使い分け

- 役割によって使い分ける
  - 開発は，主にデータサイエンティストが，SageMaker SDK を使って，Jupyter Notebook 上で行う
  - デプロイ～推論は，主にエンジニアが，AWS SDK を使って行う
- 役割の違うチームが，それぞれに合ったツールを利用



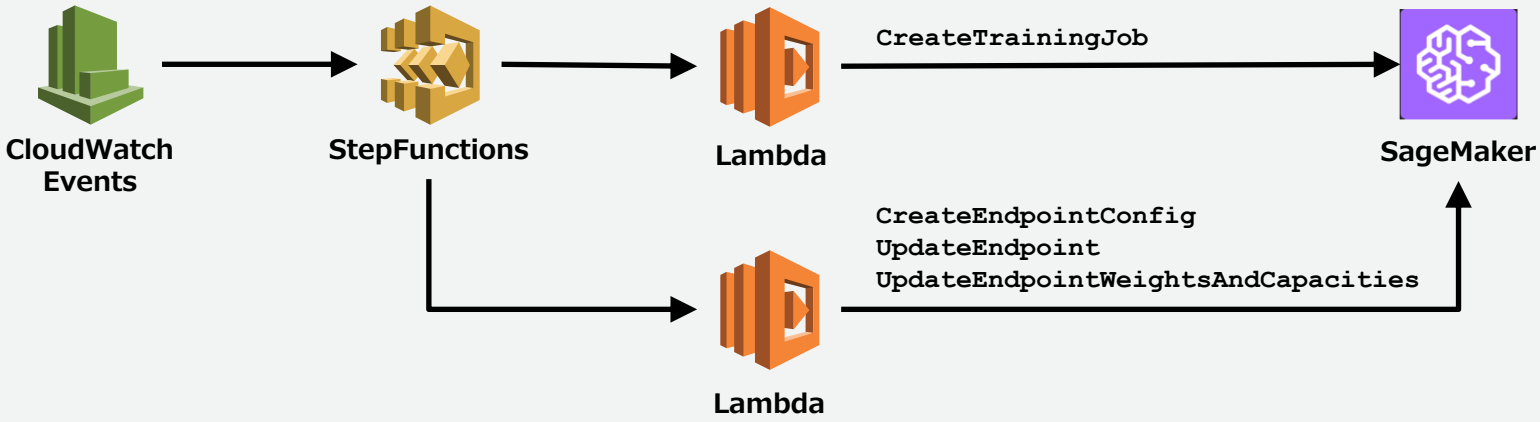
AWS SDK と SageMaker SDK の使い分け

- 役割によって使い分ける
  - 開発は，主にデータサイエンティストが，SageMaker SDK を使って，Jupyter Notebook 上で行う
  - デプロイ～推論は，主にエンジニアが，AWS SDK を使って行う
- 役割の違うチームが，それぞれに合ったツールを利用



同じジョブを最新データで定期実行

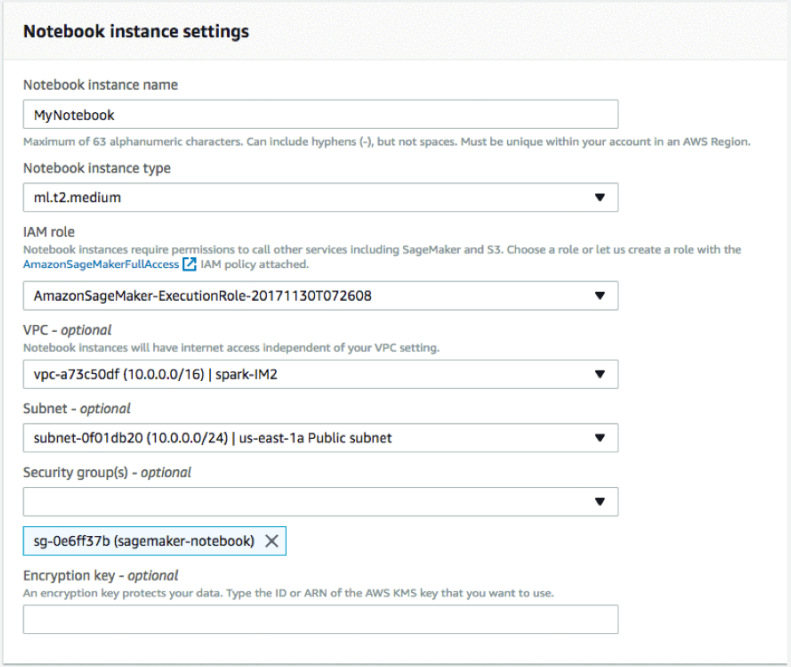
- 使用するアルゴリズムが固まって，コード自体は変えずに，新しく入ってくるデータに対して，日次でモデルを再学習
- このような場合には，基本的に SageMaker API 経由で学習ジョブの実行および，学習済モデルのデプロイを行う



開発

ノートブックから EMR への接続

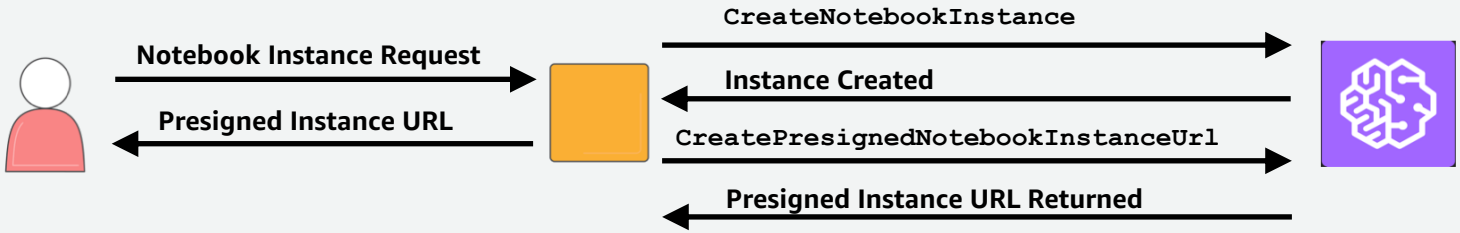
- SageMaker のノートブックから既存の EMR に接続することで，大規模データに対する前処理を高速に実行可能
- ノートブックインスタンス起動時に，EMR クラスタがあるのと同じ VPC およびサブネットを指定する必要
- ノートブックとやり取りするために，EMR クラスタには　Livy のインストールをして，適切なポートをあけておく



https://aws.amazon.com/jp/blogs/news/build-amazon-sagemaker-notebooks-backed-by-spark-in-amazon-emr/

AWS アカウントレスなノートブック環境の構築

- SageMaker には，ノートブックインスタンスにアクセス可能な Presigned URL を発行する API がある
  - `CreatePresignedNotebookInstanceUrl`
- これを活用することで，社内向けのマネージドノートブックホスティング環境を構築可能に
  - 利用者は AWS アカウントも，AWS の知識も必要としない



https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreatePresignedNotebookInstanceUrl.html

学習

分散学習

- SageMaker のビルトインアルゴリズム
  → **`instance_count`** を 2 以上にすれば，自動で分散学習

- Tensorflow/MXNet
  → **`instance_count`** に加えて，コードも分散学習に対応した
  した形にする必要あり

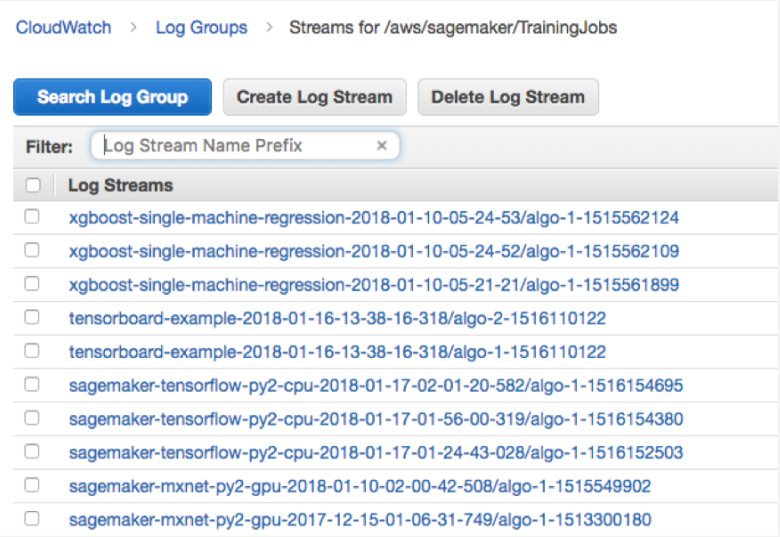
- それ以外のやり方
  → 自分ですべて記述する必要あり
  Docker コンテナ内の以下のパスに， SageMaker がホスト情報を記述
  `/opt/ml/input/config/resourceConfig.json`

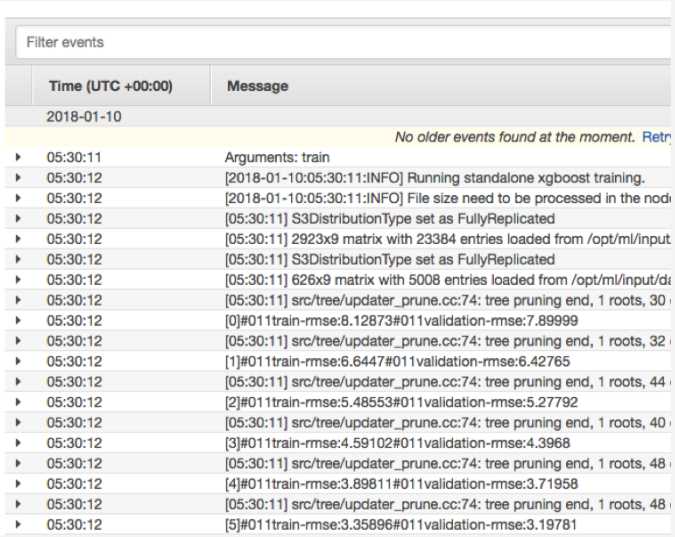
```
{
"current_host": "algo-1",
"hosts": ["algo-1","algo-2","algo-3"]
}
```

aws

学習ジョブの評価

- 学習ジョブの実行時のログは，すべて CloudWatch Logs に出力される
- 自作コンテナを使う場合は，標準出力に出したものがそのまま CloudWatch Logs に送られる
- モデル評価等は，すべてログに出力して，あとから集計





Tensorflow で学習するときのポイント

- AWS 側で Tensorflow の分散学習に最適な Docker コンテナイメージを用意
- エントリースクリプトには **model_fn** 等のインターフェースを記述
- モデル本体は，別スクリプトに切り出しておくことで，再利用性を高める
- 上記のスクリプト群は，ローカルの **source_dir** にまとめて配置
- Tensorflow バージョンは，1.4 と 1.5 に対応しており，Keras でも記述可能

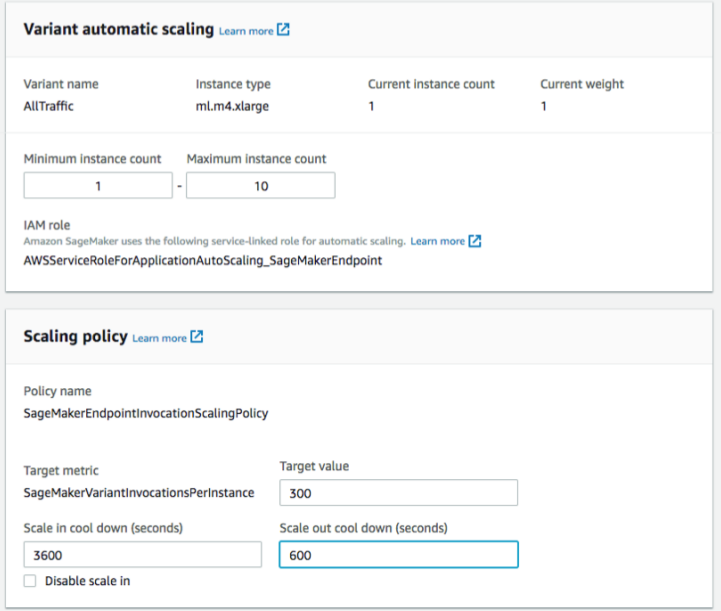
```
estimator = TensorFlow(entry_point='mnist.py',
                       source_dir='/path/to/source/code/directory/'
                       role=role,
                       framework_version='1.5',
                       train_instance_count=3,
                       train_instance_type='ml.p3.16xlarge',
                       hyperparameters={'learning_rate': 0.1},
                       output_path='s3://path/to/output/data/')
```

https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/tf-examples.html

推論

オートスケーリング

- 基本はターゲットトラッキングスケーリングポリシーを使用
- バリアント（= エンドポイントにデプロイするモデル）ごとにオートスケーリングポリシーの設定が可能
- ターゲットメトリクスは，以下の 2 種類
  - SageMakerVariantInvocationsPerInstance
    - 1 分間の 1 インスタンスあたりの平均リクエスト数
  - カスタムメトリクス
- スケールさせる最小および最大のインスタンス数，クールダウン期間も併せて指定



https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html#endpoint-auto-scaling-add-policy
https://docs.aws.amazon.com/ja_jp/autoscaling/application/userguide/application-auto-scaling-target-tracking.html

Tensorflow で推論するときのポイント

- AWS 側で Tensorflow の推論に最適な Docker コンテナイメージを用意
- 内部的には Tensorflow Serving を使用
- エントリースクリプトに，**input_fn**, **output_fn** を記述することで，推論時の前処理，後処理が可能

```
def input_fn(serialized_input, content_type):
    if content_type == "application/python-pickle":
        deserialized_input = pickle.loads(serialized_input)
        return deserialized_input
    else:
        return serialized_input
```

https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/tf-examples.html

aws

他フレームワークのモデルを ONNX 経由で MMS で推論

- AWS 側で MXNet の推論に最適な Docker コンテナイメージを用意
- SageMaker のホスティングには，Model Server for Apache MXNet (MMS)を使用
- MXNet は ONNX のモデルインポートに対応しているため，**他フレームワークで学習したモデルを ONNX 経由で MXNet モデルに変換することで，SageMaker でホスティング可能**
  - Chainer
  - PyTorch
  - Caffe2
  - Microsoft Cognitive Toolkit

A/B テスト

- 複数のモデルそれぞれに，以下のような項目を設定可能
  - インスタンスタイプ
  - インスタンス数
  - リクエスト振分の重み
- エンドポイントからのレスポンスにバリアント名が含まれるため，受け取ったアプリケーション側で，適切にログに吐き出して結果を評価

モデル名	バリアント名	インスタンスタイプ	初期インスタンス数	初期重量
linear-learner-2018-02-28-02-32-38-500	Logistic Regression	ml.m4.xlarge	3	0.8
decision-trees-sample	Decision Tree	ml.c5.9xlarge	3	0.1
sagemaker-tensorflow-py2-cpu-2018-01-22-11-49-31-334	variant-name-3	ml.p3.2xlarge	5	0.1

```
HTTP/1.1 200
Content-Type: ContentType
x-Amzn-Invoked-Production-Variant: InvokedProductionVariant

Body
```

https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/API_runtime_InvokeEndpoint.html

aws

その他

セキュリティ関連

- 学習と推論のジョブにおいて，オプションパラメータとして KMS key ID を指定することで，SSE-KMS を利用可能
  - `CreateTrainingJob`
  - `CreateEndpointConfig`
- 以下のものをすべて暗号化可能
  - 学習時の入出力データ
  - 学習用インスタンスのストレージ
  - エンドポイントインスタンスのストレージ
- Cloudtrail に対応済み
- PCI DSS コンプライアンスに対応済み

https://aws.amazon.com/about-aws/whats-new/2018/01/aws-kms-based-encryption-is-now-available-in-amazon-sagemaker-training-and-hosting/
https://aws.amazon.com/about-aws/whats-new/2018/01/aws-cloudtrail-integration-is-now-available-in-amazon-sagemaker/
https://aws.amazon.com/about-aws/whats-new/2018/01/amazon-sagemaker-achieves-pci-dss-compliance/

価格

- オンデマンド ML インスタンス
  - SageMaker の開発・学習・推論の各パートごとに，利用したインスタンスの料金が，従量課金として請求される（最低実行時間 1 分間）
- ML 汎用ストレージ
  - インスタンスにアタッチしたストレージの料金
  - バージニア北部リージョンで，0.14 USD/GB/月
- データ処理量
  - 各インスタンスに対する入出力データの量に応じて課金
  - バージニア北部リージョンで，0.016 USD/GB

https://aws.amazon.com/jp/sagemaker/pricing/

リファレンス

さまざまな情報が，以下の 3 箇所にまとまっている

- SageMaker Example Notebooks
  - https://github.com/awslabs/amazon-sagemaker-examples
- SageMaker SDK
  - https://github.com/aws/sagemaker-python-sdk
    (Doc はこちら: https://readthedocs.org/projects/sagemaker/)
- SageMaker 公式ドキュメント
  - https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/whatis.html

ビルトインアルゴリズムのノートブック

各アルゴリズムについて，それぞれひとつずつサンプル実行を行なったノートブックがある．また公式ドキュメントに，詳細な仕様も記載

https://github.com/awslabs/amazon-sagemaker-examples/tree/master/introduction_to_amazon_algorithms
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/algos.html

- Linear Learner
- Factorization Machines
- XGBoost
- Image Classification
- seq2seq
- K-means
- PCA
- LDA
- Neural Topic Model
- DeepAR Forecasting *New!*
- BlazingText (word2vec) *New!*

Tensorflow のノートブック

- Tensorflow の使用例と記述の説明
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_abalone_age_predictor_using_keras/tensorflow_abalone_age_predictor_using_keras.ipynb
- Tensorflow で Tensorboard を使う
  - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/tensorflow_resnet_cifar10_with_tensorboard
- Tensorflow で分散学習を行う
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_distributed_mnist/tensorflow_distributed_mnist.ipynb
- Keras + Tensorflow の使用例
  - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/tensorflow_abalone_age_predictor_using_keras
- そのほかに，MXNet についても以下に同様にまとまっている
  - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk

それ以外のやり方のノートブック

- scikit-learn の学習・ホスティングを SageMaker で行う
  （SageMaker 用のコンテナイメージを作成する手順も併せてまとまっている）
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb
- SageMaker で R を使う
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/install_r_kernel/example_r_notebook.ipynb
- SageMaker で暗号化データを使う
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/handling_kms_encrypted_data/handling_kms_encrypted_data.ipynb
- 別のところで作った XGBoost モデルを使って SageMaker でホスティング
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/xgboost_bring_your_own_model/xgboost_bring_your_own_model.ipynb
- 別のところで作った　Tensorflow モデルを使って　Sagemaker でホスィング
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/tensorflow_iris_byom/tensorflow_BYOM_iris.ipynb
- そのほかに，Redshift との連携や MXNet・R などのモデル持ち込みも
  - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality

オンラインセミナー資料の配置場所

AWS クラウドサービス活用資料集

- http://aws.amazon.com/jp/aws-jp-introduction/

AWS Solutions Architect ブログ

- 最新の情報、セミナー中のQ&A等が掲載されています
- http://aws.typepad.com/sajp/

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、
お得なキャンペーン情報などを日々更新しています！

AWSの導入、お問い合わせのご相談

AWSクラウド導入に関するご質問、お見積り、資料請求を
ご希望のお客様は以下のリンクよりお気軽にご相談ください
https://aws.amazon.com/jp/contact-us/aws-sales/

※「AWS 問い合わせ」で検索してください

aws