Amazon SageMaker

Amazon Web Services Japan, K. K.

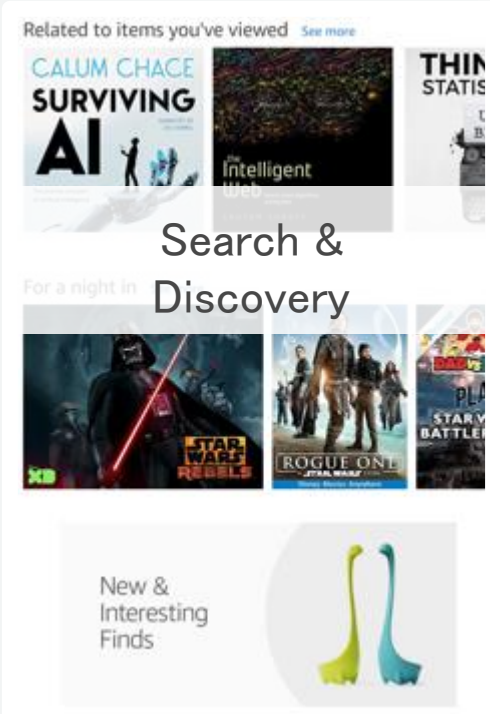
2018/12/1

本日のアジェンダ

- AWS ML サービス概要　…5分
- SageMaker ご紹介　…35分
- SageMaker ハンズオン　…35分
- MLサービス最新Update (Re:Inventでの発表分)　…5分

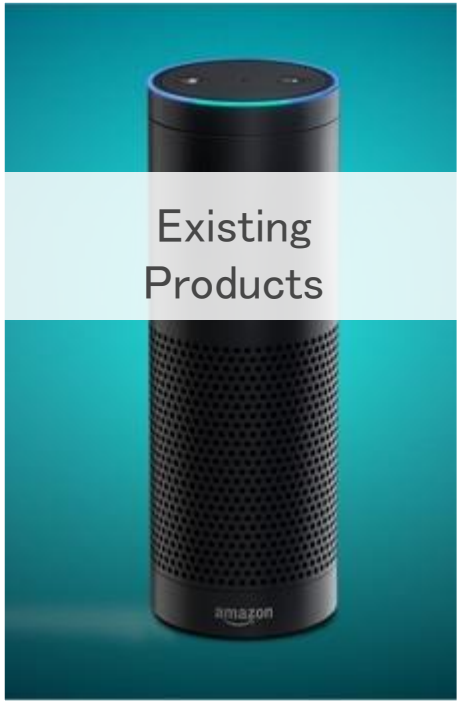
AWS ML サービス概要

過去20年間にわたる継続的な AI への投資

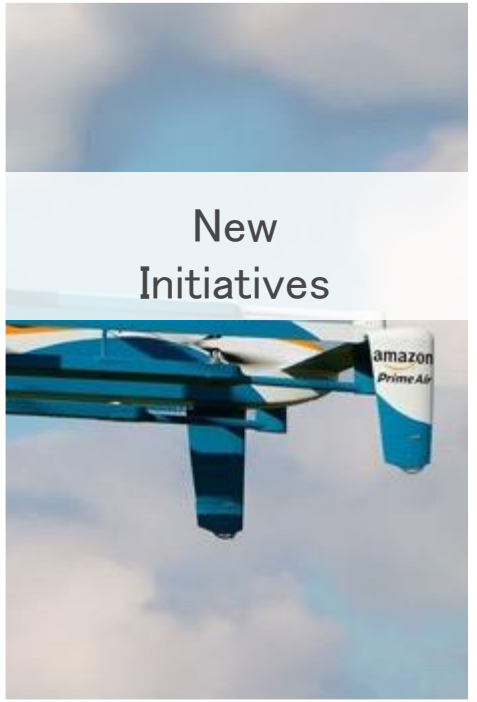


Search & Discovery

Fulfilment & Logistics

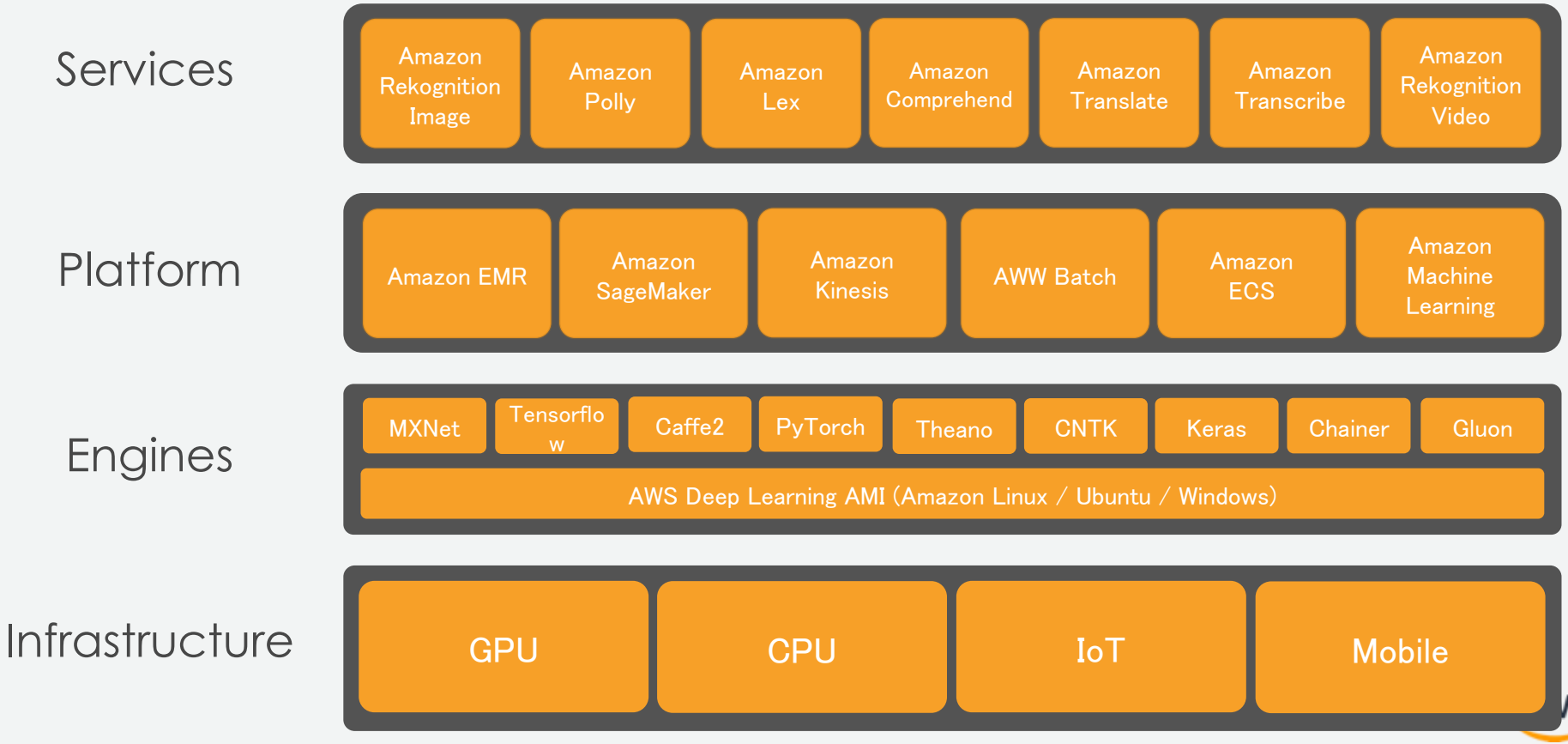


Existing Products



New Initiatives

AWS が提供する ML サービススタック



100を超えるAWSサービス群

コンピュート

EC2 Lambda EC2 Container Service Elastic Beanstalk Elastic Load Balancing

ネットワーク

VPC Direct Connect Route 53

アナリティクス

Athena EMR Data Pipeline Kinesis Machine Learning QuickSight Elasticsearch Service

AI

Lex Machine Learning Polly Rekognition

開発ツール

CodeBuild CodeCommit CodeDeploy CodePipeline

管理ツール

CloudWatch Cloud Formation CloudTrail Config OpsWorks Service Catalog

セキュリティ

Identity & Access Management Directory Service Trusted Advisor Cloud HSM Key Management Service Web App Firewall

ストレージ & 配信

S3 CloudFront EFS Glacier Storage Gateway Snowball

アプリケーションサービス

API Gateway AppStream CloudSearch Elastic Transcoder Step Functions SES SQS SWF

Hubs

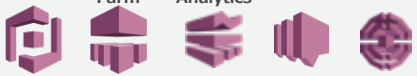
Mobile Hub

ゲーム

GameLift

モバイルサービス

Cognito Device Farm Mobile Analytics SNS Pinpoint



データベース

RDS DynamoDB ElastiCache RedShift Simple DB Database Migration Service

IoT

IoT

エンタープライズアプリ

WorkSpaces WorkDocs WorkMail

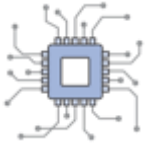
aws

SageMaker ご紹介

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

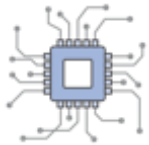
大量（～数百台）の GPU または CPUインスタンス
継続的なデプロイ
さまざまなデバイスで動作

機械学習の流れ

開発 & 学習

学習に使うコードを記述
小規模データで動作確認

大量の GPU
大規模データの処理
試行錯誤の繰り返し



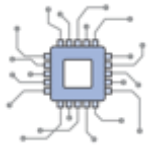
推論

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

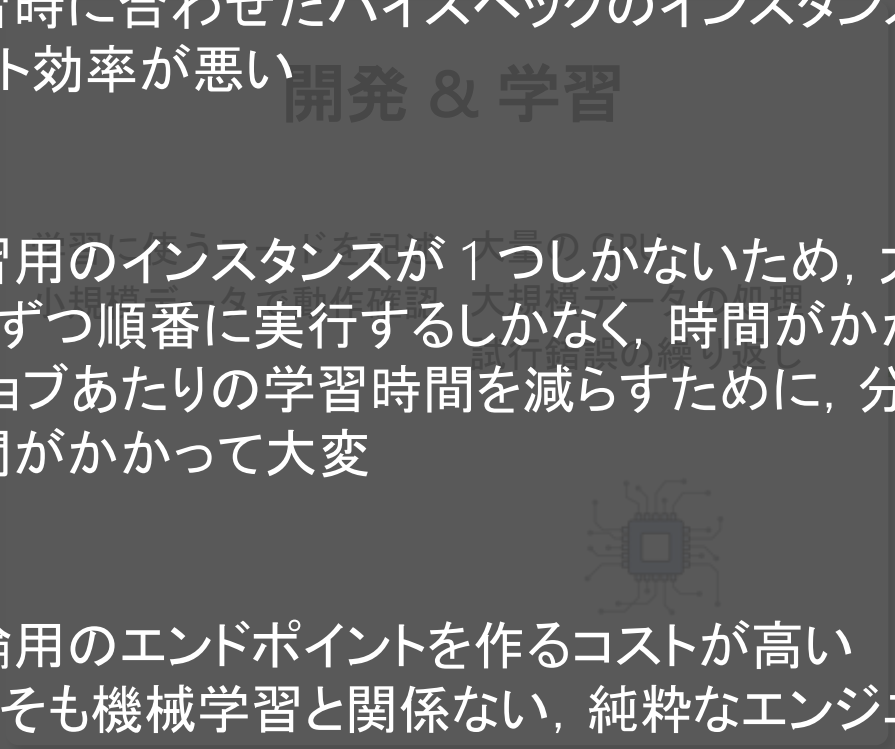
- 学習時に合わせたハイスペックのインスタンスで開発もするため，コスト効率が悪い

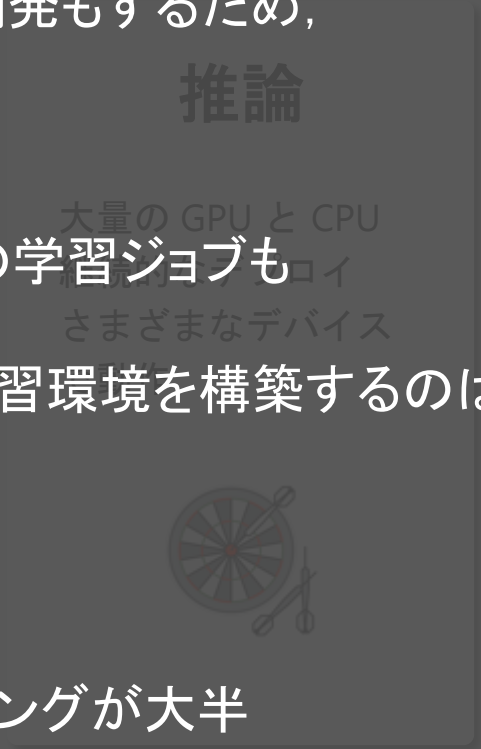
学習

- 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

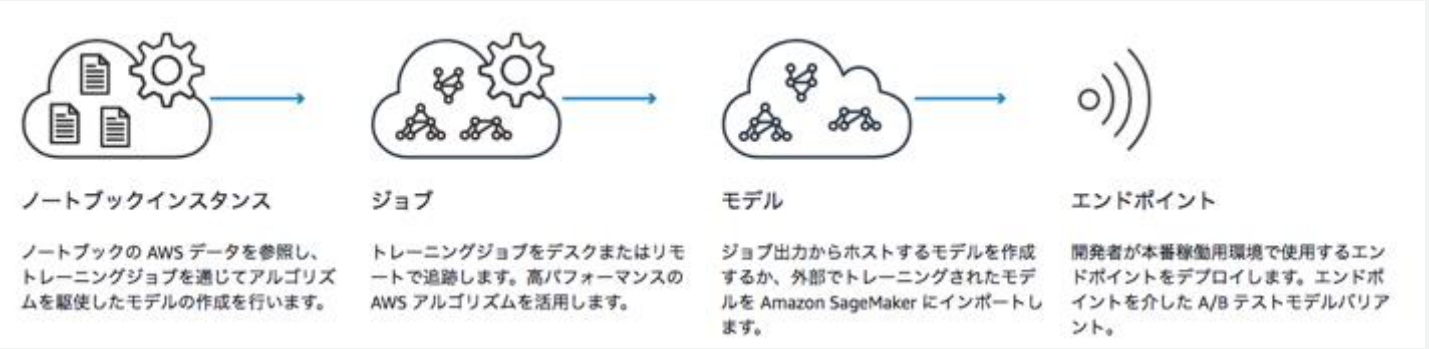




SageMaker の概要

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，カリフォルニア北部，モントリオール，アイルランド，フランクフルト，ロンドン，**東京**，ソウル，シドニー，シンガポール，ムンバイで提供



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

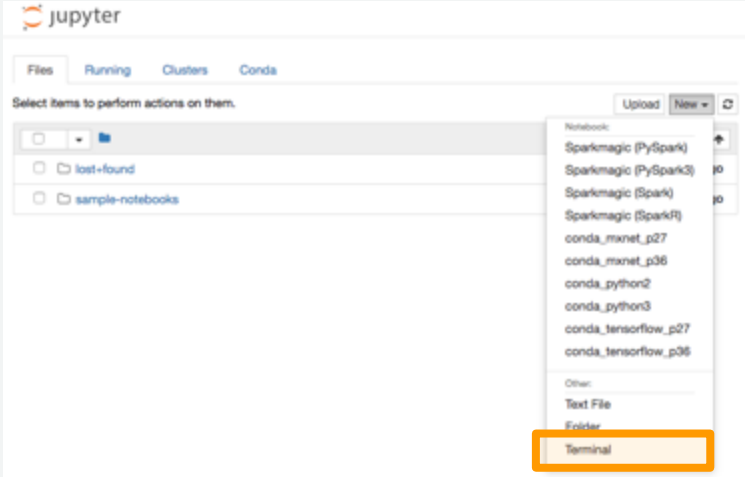
推論

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 ml.t2 / ml.m4 / ml.p2 / ml.p3 の 4 種類
  - インスタンスはお客さま VPC の外側に存在するが，ENI 経由で VPC 内リソースにアクセスさせることも可能
  - アタッチするストレージは 5GB から 16TB まで 1GB 単位で指定可能

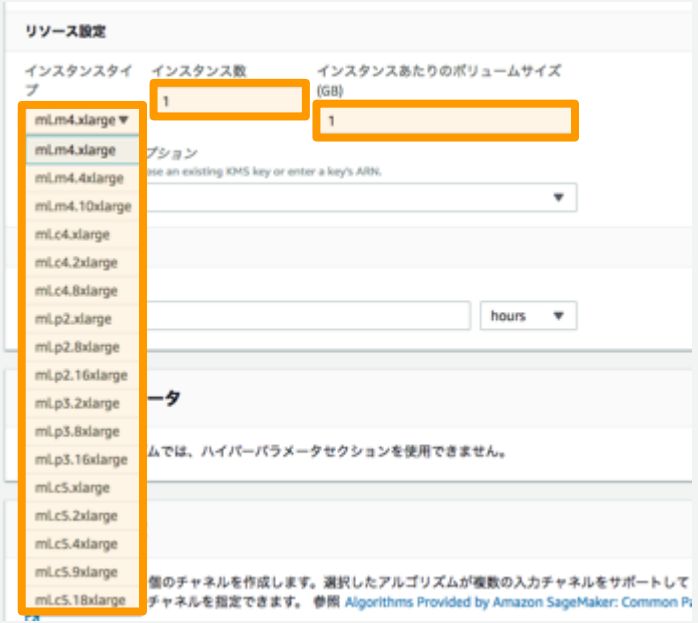


* あくまで開発環境として使うため，可能な範囲で小さなインスタンスを選択するのがおすすめ

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

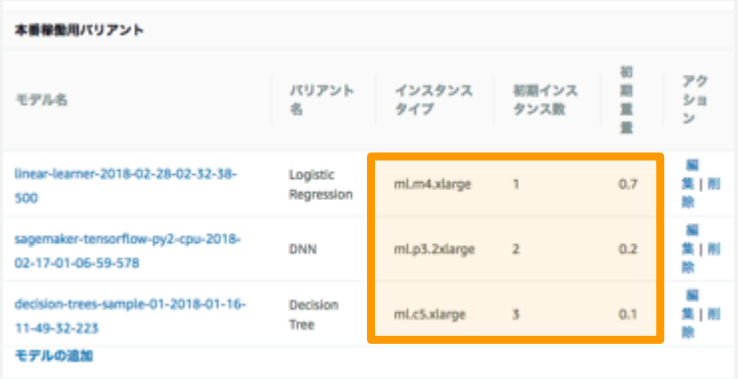
  - コンテナを動かすインスタンスタイプ，インスタンス数，各インスタンスのボリュームサイズを指定可能
  - インスタンス数を 2 以上にすれば，自動で分散学習を実行
  - インスタンスタイプは m4, m5, c4, c5, p2, p3 から選択可能
  - 入力データの読み込み元と，学習済モデルの出力先は，いずれも S3



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/Chainer/PyTorch/MXNet 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/Chainer/PyTorch/MXNet 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

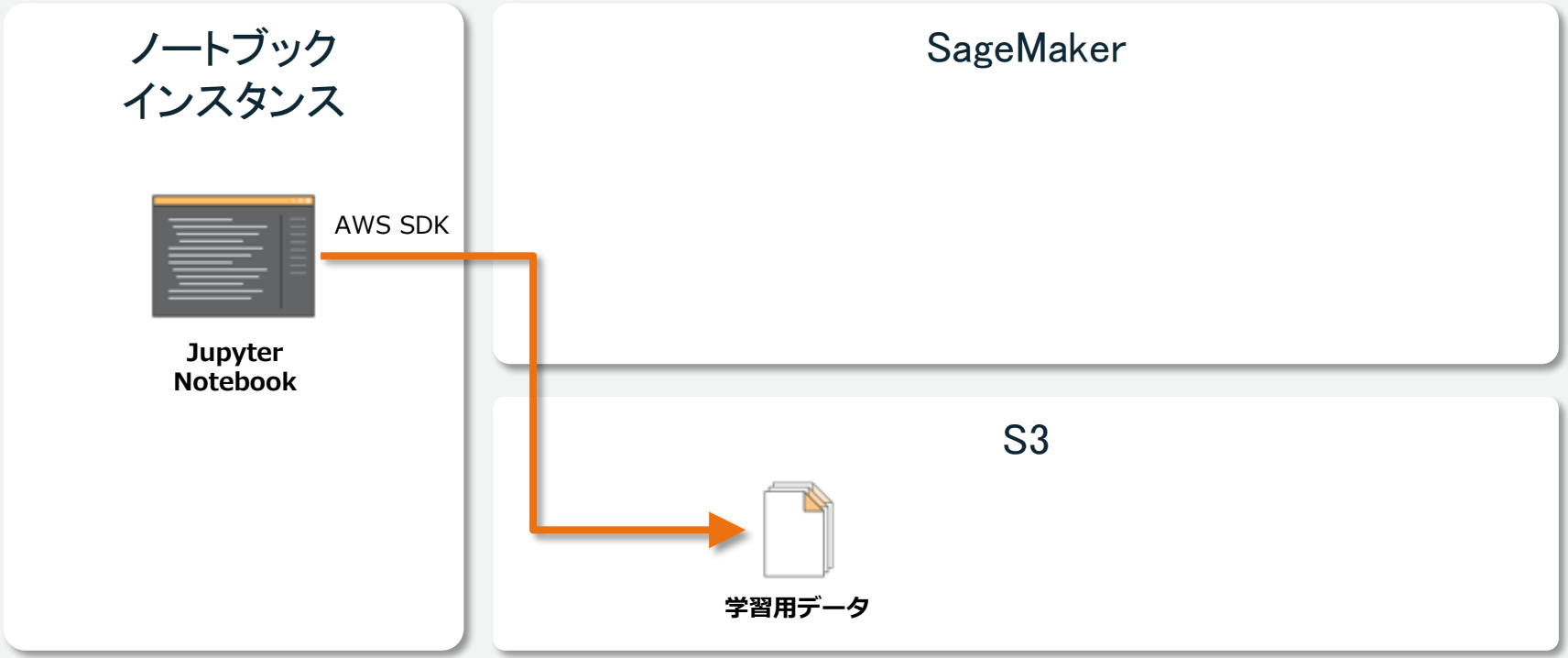
- Linear Learner
- Factorization Machines
- XGBoost
- Image Classification
- seq2seq
- K-means
- k-NN
- Object2Vec *New!*
- PCA
- LDA
- Neural Topic Model
- DeepAR Forecasting
- BlazingText (word2vec)
- Random Cut Forest
- Object Detection
- IP Insights *New!*

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

aws

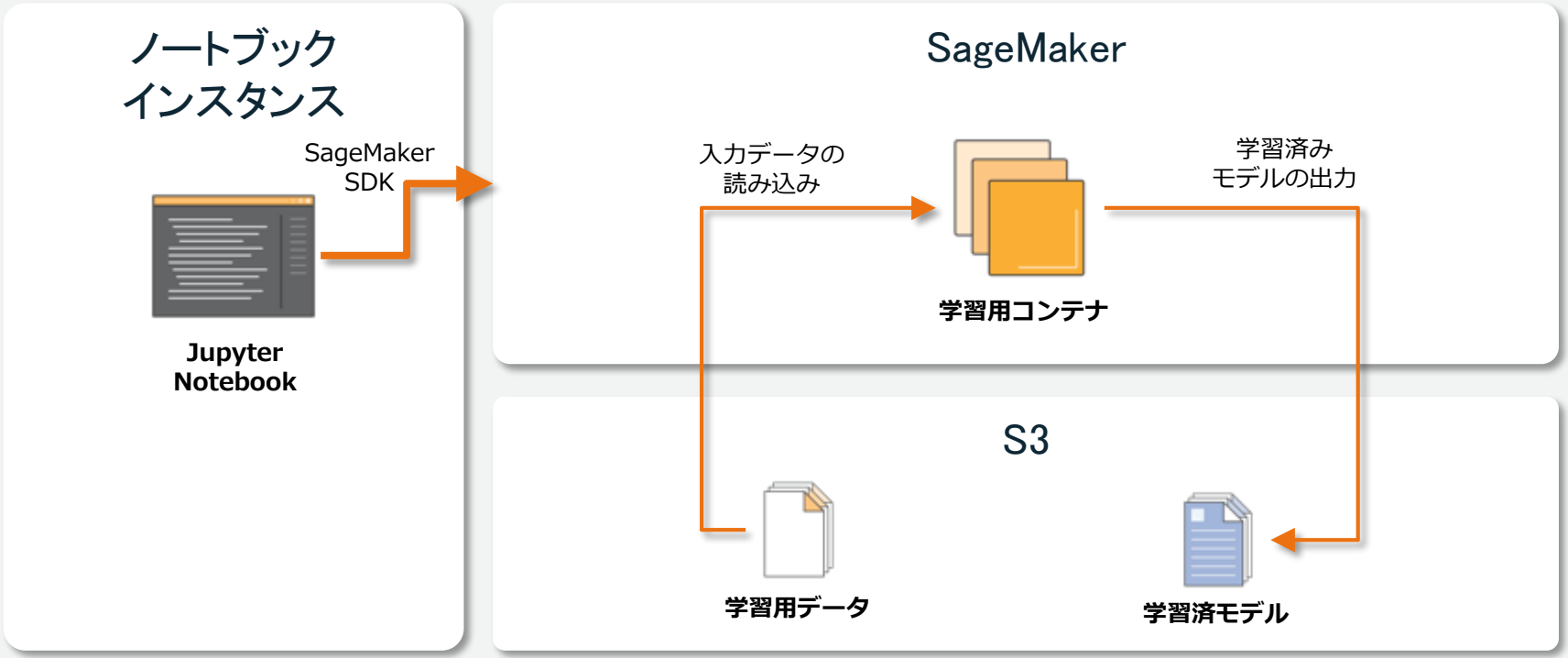
開発

ノートブックから，**学習用データ**を作成して，S3 に置く



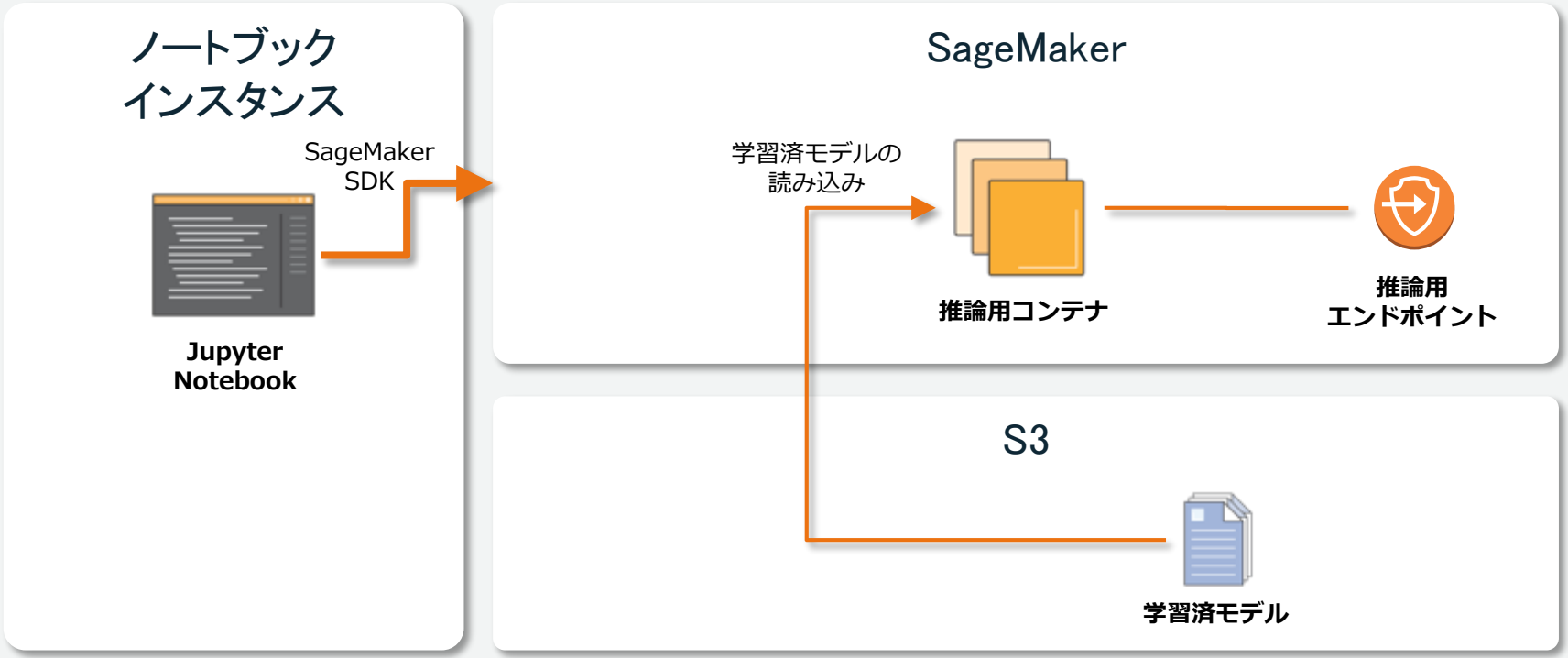
学習

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/Chainer/PyTorch/MXNet 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

Chainer を使うときに必要なインタフェース

学習用コード

- **`__main__`**: main 関数内に，Chainer のモデルを記述して，run() を実施し，最後に存する処理までを記述．環境変数経由で，GPU 数や入力データのディレクトリ，の出力場所等を取得可能

推論用コード

- **`model_fn:`** 学習済みモデルのロード処理を記述
- そのほか，推論時の前処理，後処理も記述可能

https://docs.aws.amazon.com/sagemaker/latest/dg/chainer.html
https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/chainer_mnist/chainer_mnist_single_machine.py

aws

PyTorch を使うときに必要なインタフェース

学習用コード

- **`__main__`**: main 関数内に，PyTorch のモデルを記述して，run() を実施し，最後存する処理までを記述．環境変数経由で，GPU 数や入力データのディレクトリ，の出力場所等を取得可能

推論用コード

- **`model_fn:`** 学習済みモデルのロード処理を記述
- そのほか，推論時の前処理，後処理も記述可能

https://docs.aws.amazon.com/sagemaker/latest/dg/pytorch.html
https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst

aws

MXNet (1.3-) を使うときに必要なインタフェース

学習用コード

- **`__main__`**: main 関数内に，MXNet のモデルを記述して，fit() を実施し，最後にする処理までを記述．環境変数経由で，GPU 数や入力データのディレクトリ，出力場所等を取得可能

推論用コード

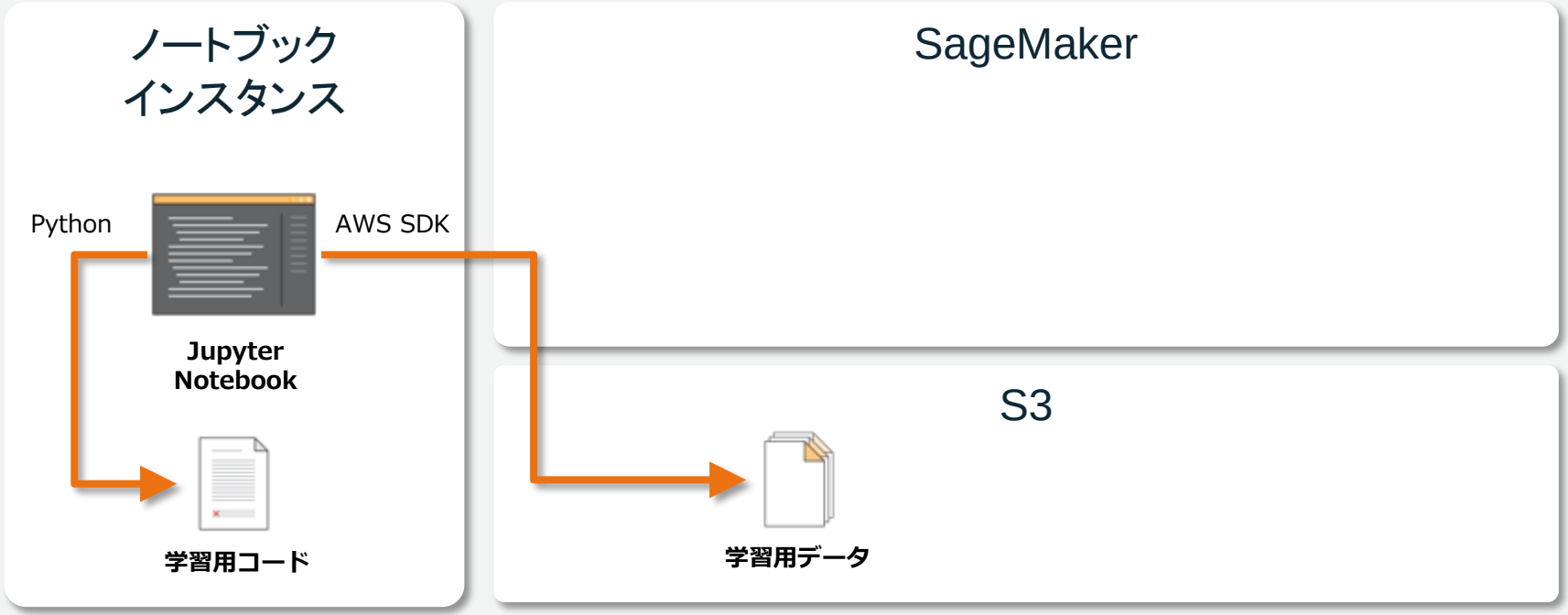
- **`model_fn:`** 学習済みモデルのロード処理を記述
- **`transform_fn:`** リクエストデータの予測処理を記述
- そのほか，Gluon 用インタフェース，Module モデル用インタフェースも定義することが可能

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

aws

開発

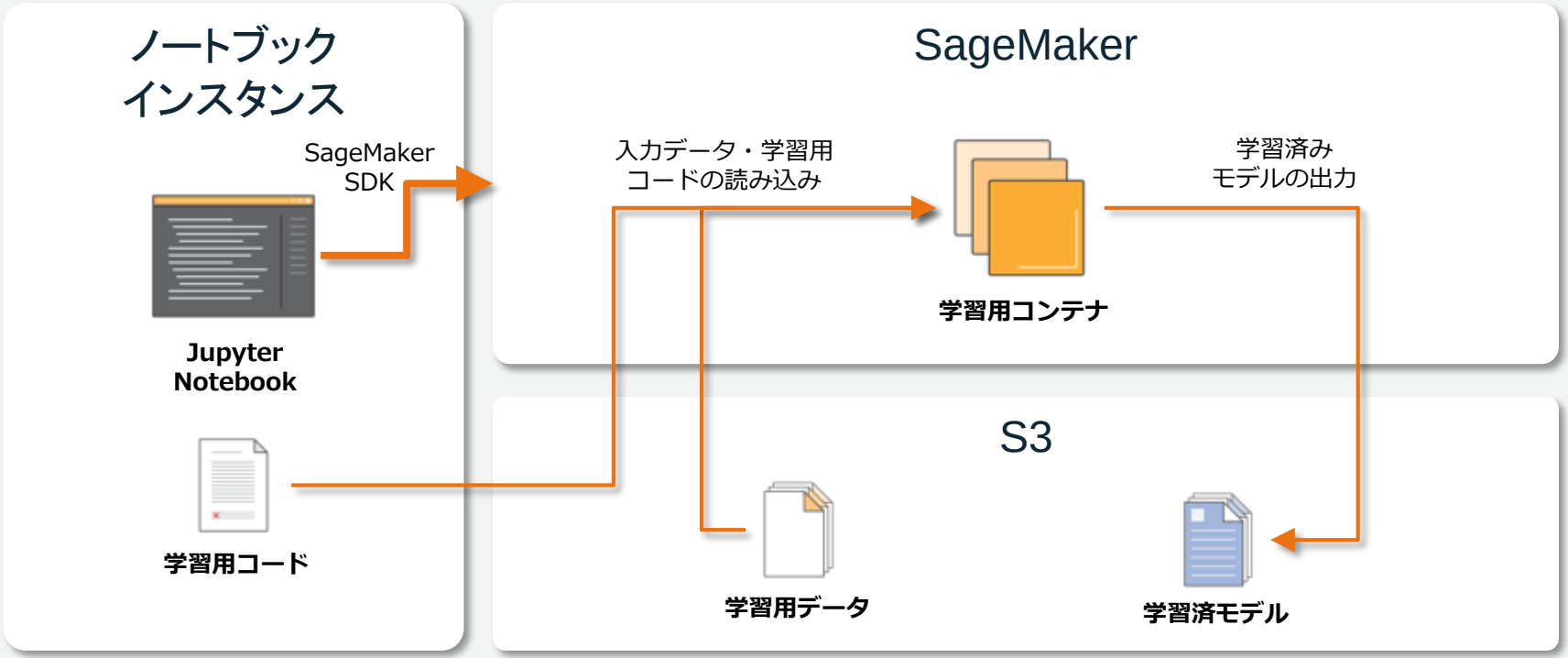
ノートブックで，**学習用コード**を作成してローカルに置き*，
学習用データを作成してS3 に置く



* 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

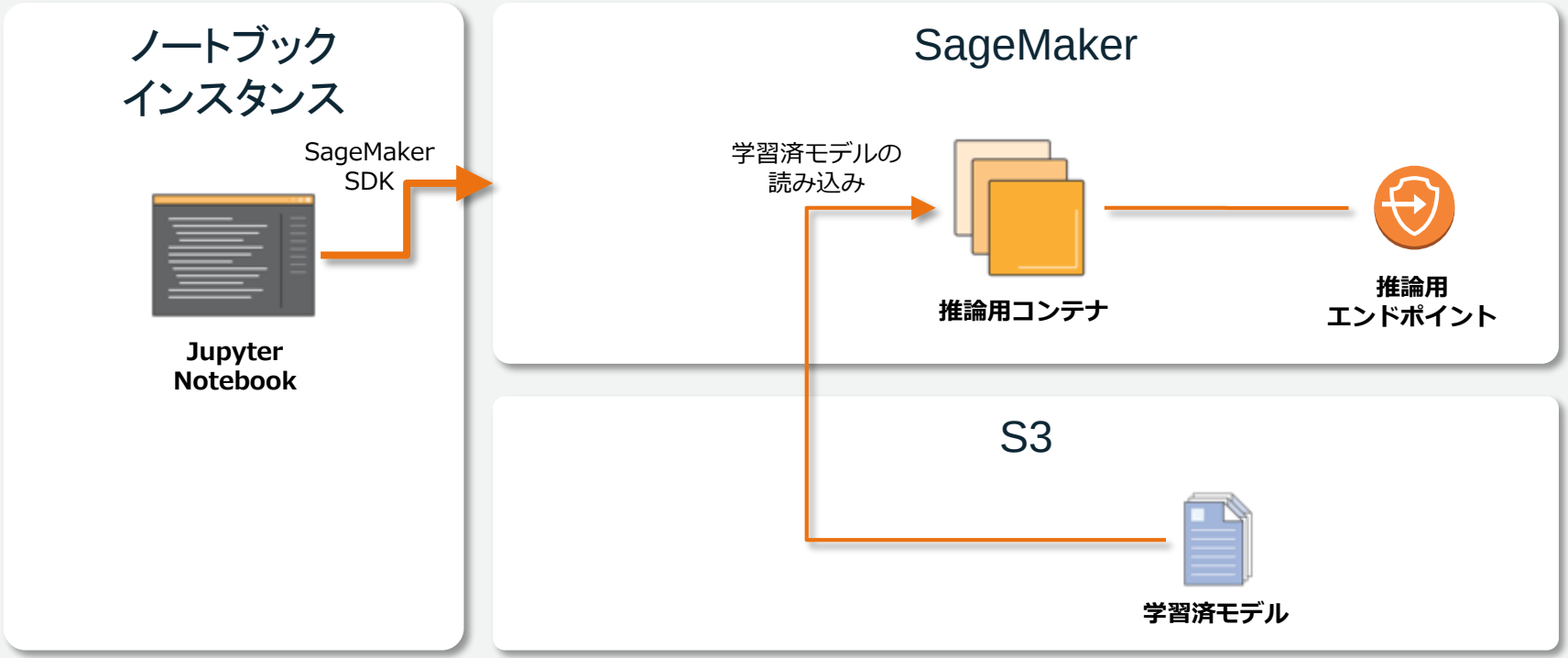
学習

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

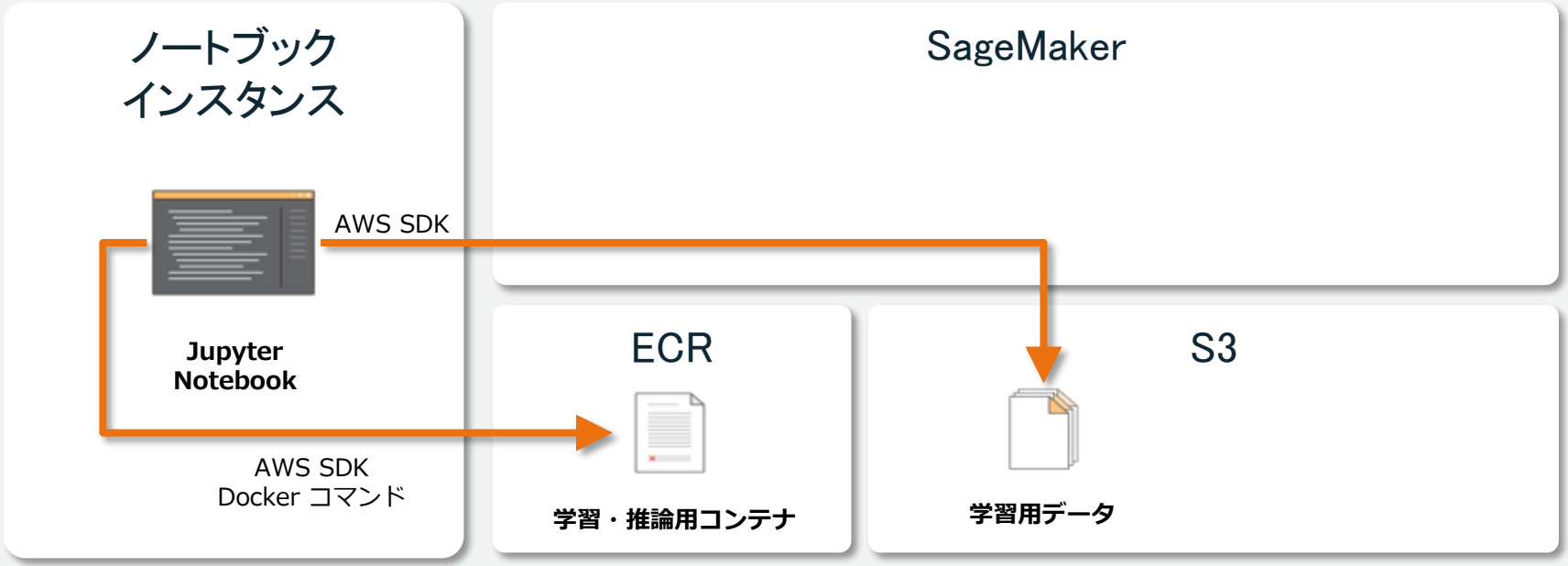


```
from sagemaker.tensorflow import TensorFlow

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')
```

それ以外のやり方で開発を行う

開発

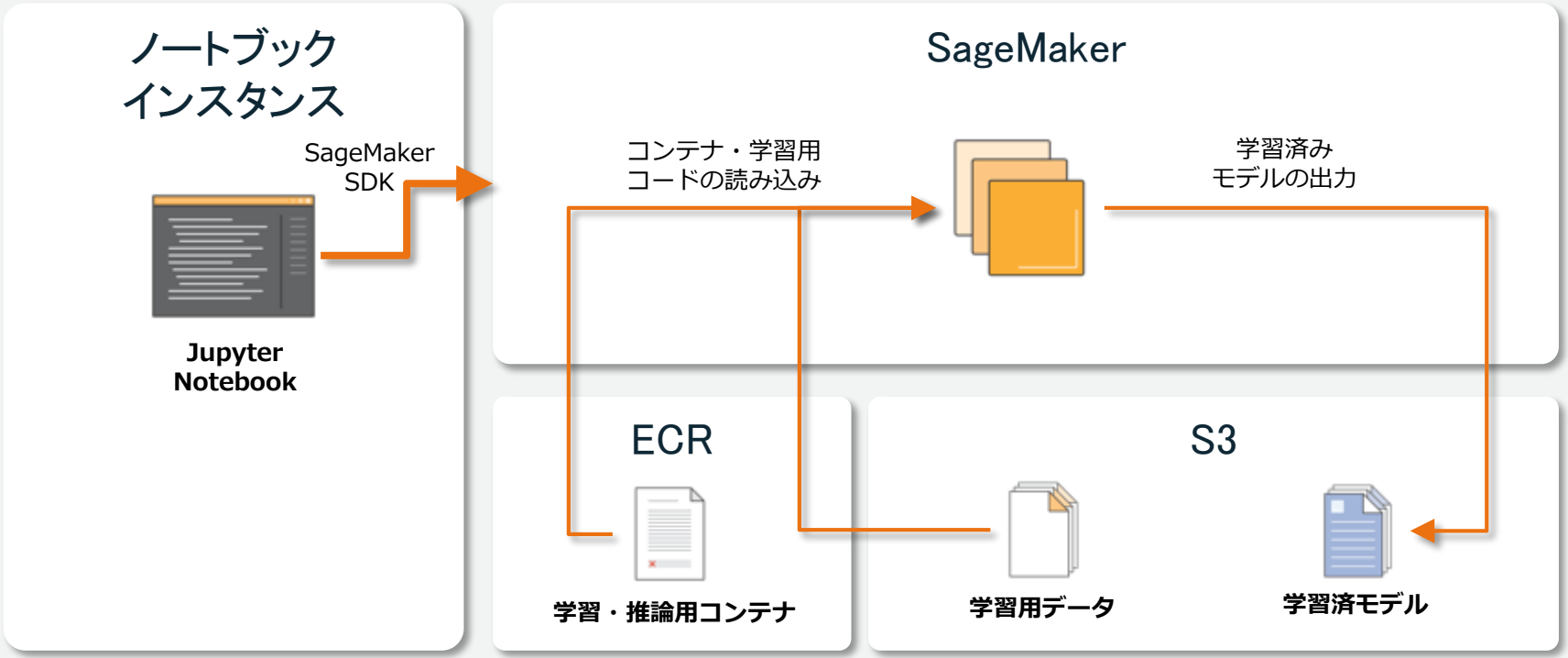
ノートブックで，**学習コードと推論用 Web サーバが入ったコンテナ**を作成して
ECR* に **push** し，**学習用データ**を作成してS3 に置く



* Elastic Container Registry

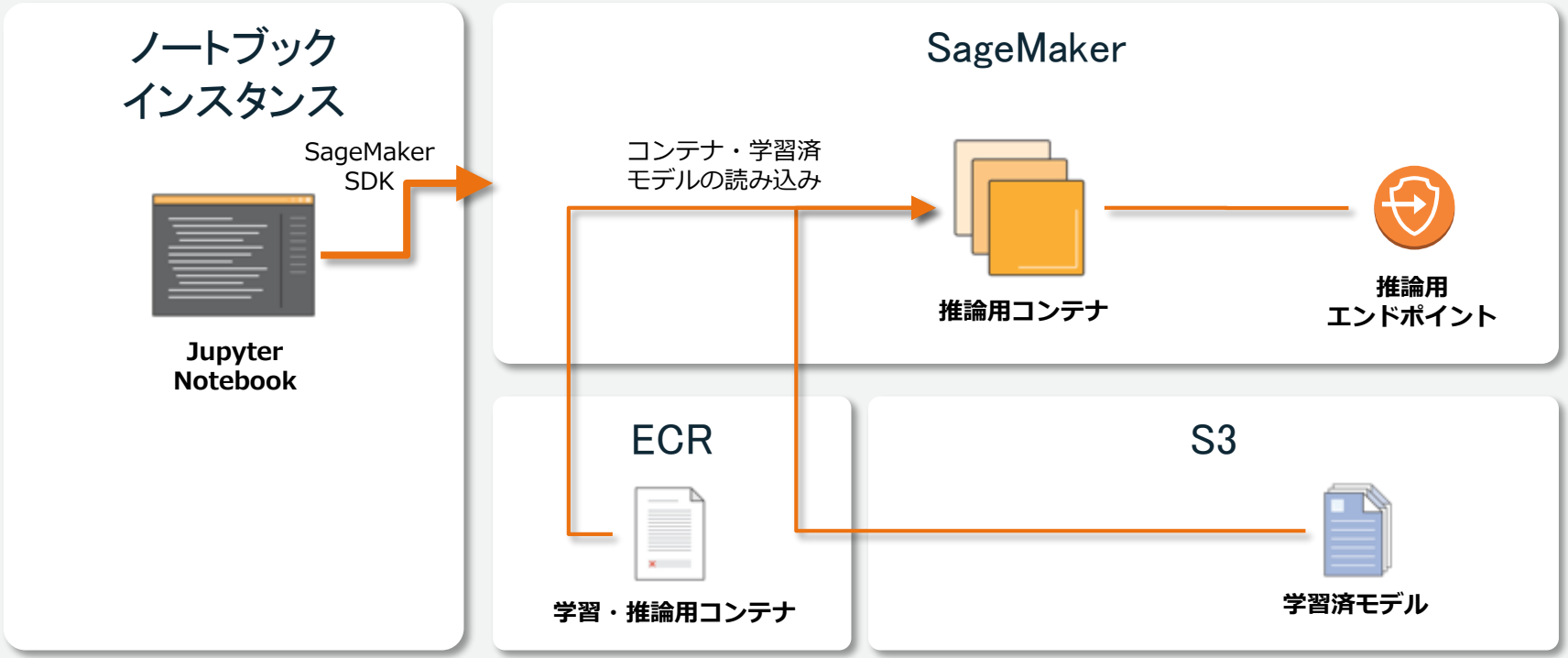
学習

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')
```

実践的な活用法

SageMaker の開発・デプロイフロー

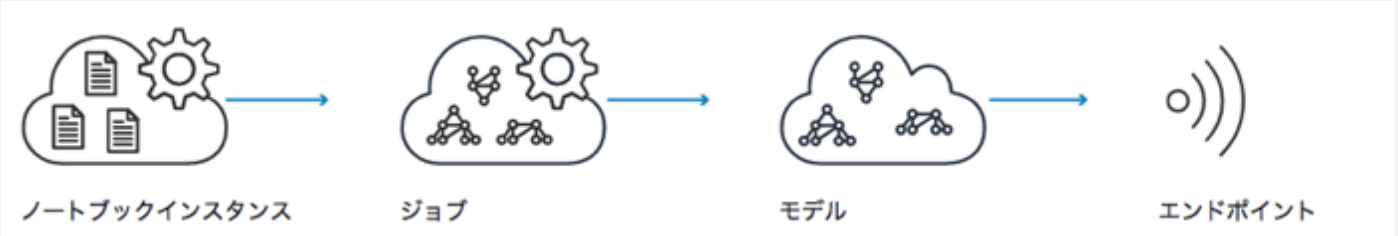
SageMaker の 3 要素は，それぞれ個別で利用可能

- 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

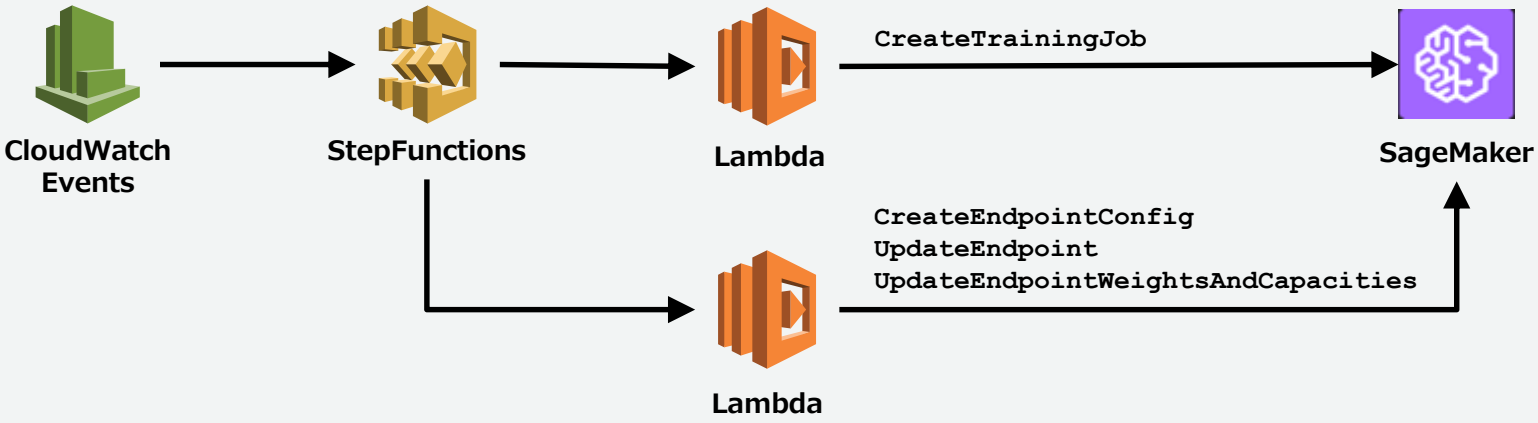
AWS SDK と SageMaker SDK の使い分け

- 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



同じジョブを最新データで定期実行

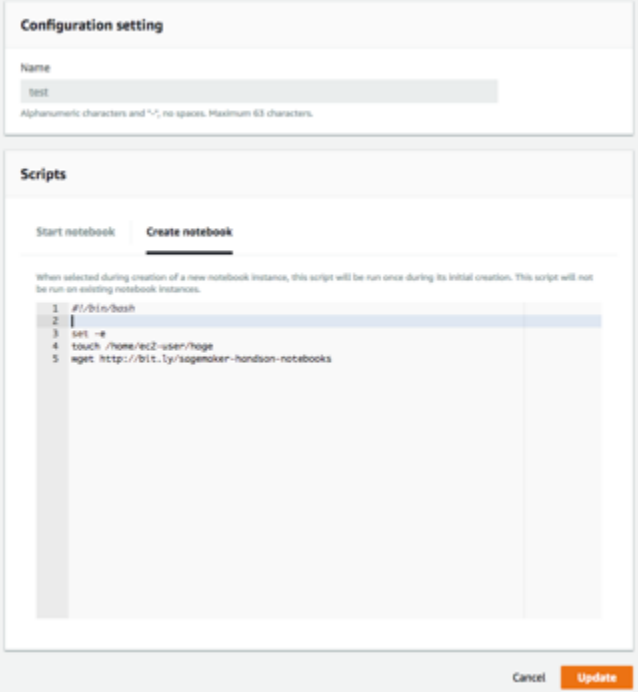
- 使用するアルゴリズムが固まって，コード自体は変えずに，新しく入ってくるデータに対して，日次でモデルを再学習
- このような場合には，基本的に SageMaker API 経由で学習ジョブの実行および，学習済モデルのデプロイを行う



開発

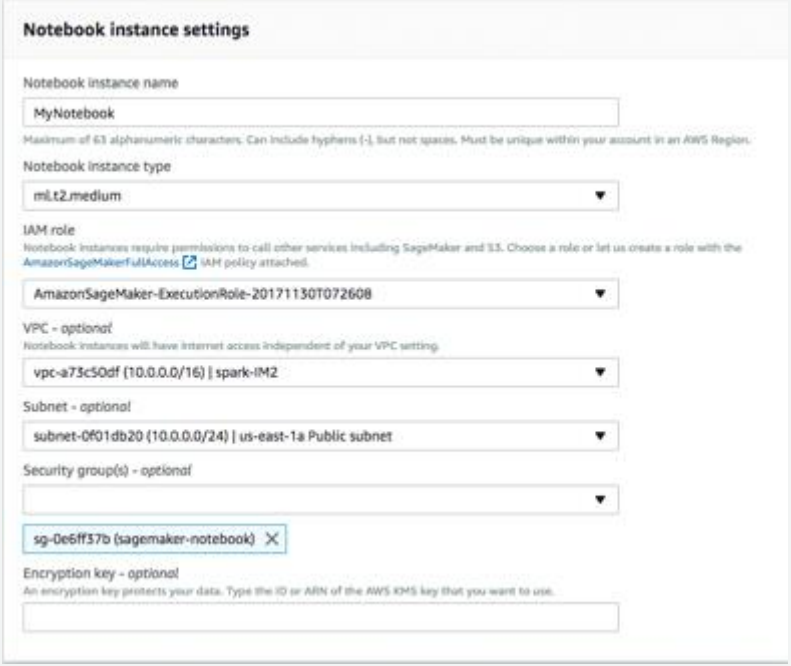
ライフサイクル設定とインターネットアクセス

- インスタンス起動時と開始時のセットアップ処理を自動化可能
  - 開始時は，インスタンスを停止→再開した際にも実行される
  - インスタンス作成時に，ライフサイクル設定を付与しておくだけ
  - **`UpdateNotebookInstance`** API 経由で，新しいライフサイクル設定を反映させることも可能
- VPC にアタッチしてインスタンスを作成する場合，外向きのインターネットアクセスを禁止することが可能



ノートブックから EMR への接続

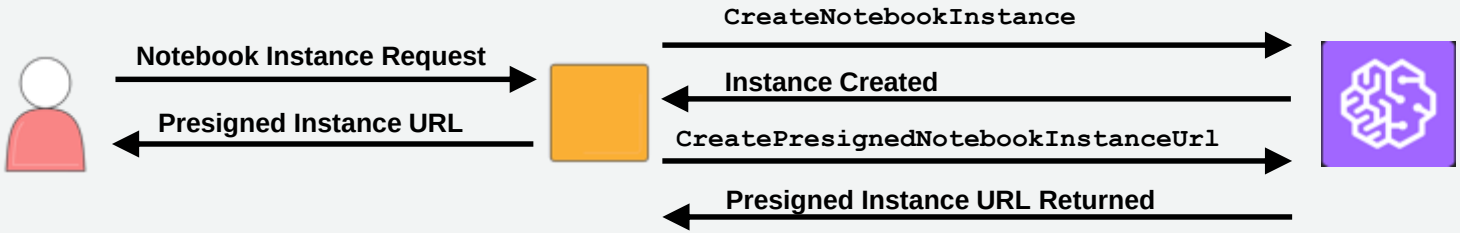
- SageMaker のノートブックから既存の EMR に接続することで，大規模データに対する前処理を高速に実行可能
- ノートブックインスタンス起動時に，EMR クラスタがあるのと同じ VPC およびサブネットを指定する必要
- ノートブックとやり取りするために，EMR クラスタには Livy のインストールをして，適切なポートをあけておく



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/Chainer/PyTorch/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"]
}
```

ハイパーパラメータのチューニング

- **Estimater** の初期化時に **hyperparameters** で引き渡すパラメタに関して，ベイズ最適化によるパラメタの自動チューニングを実行可能
- ビルトイン，Tensorflow/Chainer/PyTorch/MXNet に加え，BYOA でも利用可能
- ターゲットメトリクスも自由に指定可能
- 前に実行したジョブの結果を引き継ぐ形の，チューニングジョブのウォームスタートにも対応

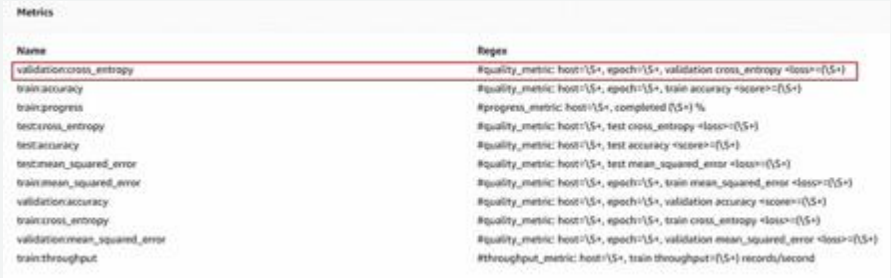
```
tuner = HyperparameterTuner(estimator,
                            objective_metric_name,
                            hyperparameter_ranges,
                            metric_definitions,
                            objective_type='Minimize',
                            max_jobs=9,
                            max_parallel_jobs=3,
                            base_tuning_job_name='hpo-r-byoa-XX')

tuner.fit({'train': 's3://{}/{}/train'.format(bucket, prefix)})
```

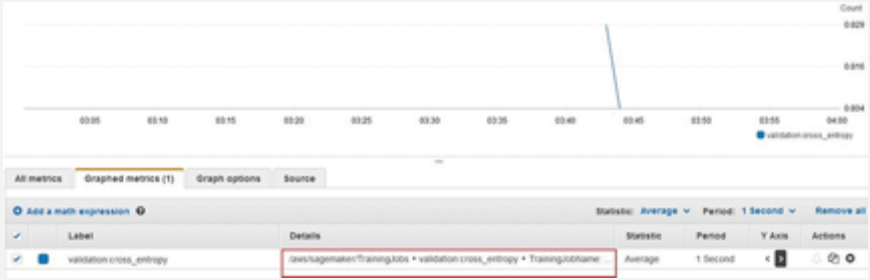
https://aws.amazon.com/jp/blogs/machine-learning/amazon-sagemaker-automatic-model-tuning-becomes-more-efficient-with-warm-start-of-hyperparameter-tuning-jobs/
https://github.com/awslabs/amazon-sagemaker-examples/tree/master/hyperparameter_tuning
https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning

aws

学習ジョブの評価の可視化

- CloudWatch Metrics 経由で，学習ジョブの指標を可視化することが可能
- **`CreateTraininJob API`** 実行時に，標準出力で出されるログに対する正規表現を指定することで，任意のメトリクスをログから取得して可視化することが可能
- ビルトインアルゴリズムは，初めからvalidation:cross_entropy のようなメトリクスに対応している

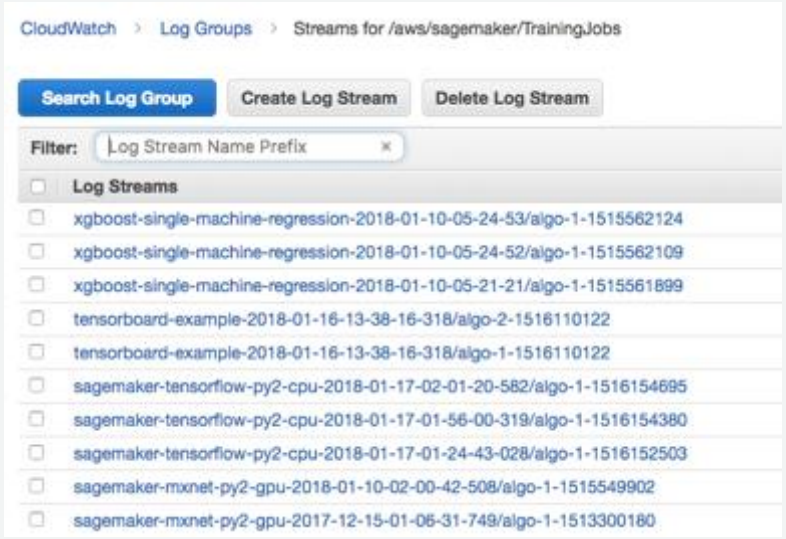


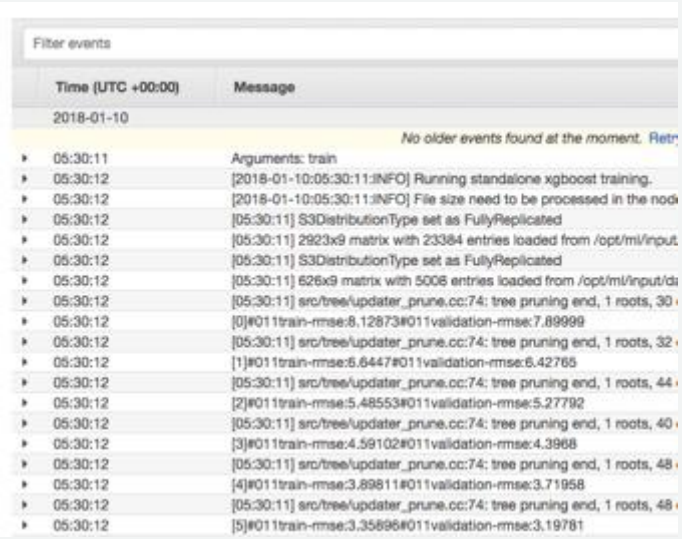


https://aws.amazon.com/jp/blogs/machine-learning/easily-monitor-and-visualize-metrics-while-training-models-on-amazon-sagemaker/

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，1.4-1.10 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

aws

ローカルでのテスト

- SageMaker で使用している Tensorflow/Chainer/PyTorch/MXNet のコンテナは github に公開されており，ダウンロードが可能
  - SageMaker 上で実行する前に，ローカルにコンテナを pull してきて，ジョブ / エンドポイントの動作テストを実行することが可能
  - 中身を確認もできるし，自分用のカスタムコンテナを作成することも可能
- ローカルテストは，**`instance_type` を `'local'` とするだけ**

```
mnist_estimator = TensorFlow(entry_point='mnist.py',
                             role=role,
                             training_steps=10,
                             evaluation_steps=10,
                             train_instance_count=2,
                             train_instance_type='local')
```

```
mnist_predictor = mnist_estimator.deploy(initial_instance_count=1,
                                         instance_type='local'
```

https://github.com/aws/sagemaker-python-sdk#local-mode
https://github.com/aws/sagemaker-tensorflow-containers
https://github.com/aws/sagemaker-mxnet-containers
https://github.com/aws/sagemaker-chainer-container
https://github.com/aws/sagemaker-pytorch-container

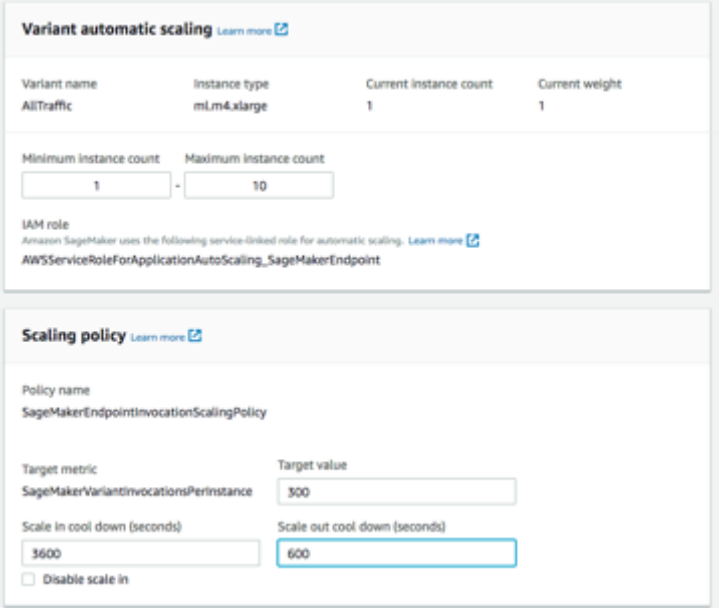
PIPE モードによるデータの高速な読み込み

- 学習時のデータ読み込み方法は，以下の 2 種類がある
  - FILE: 学習用のデータをすべて学習インスタンスにコピー
  - PIPE: 学習用のデータを，必要なタイミングで必要なぶんだけ S3 API 経由でストリームとして取得
- 以下の場合は，PIPE モードを使うことでパフォーマンスアップが期待できる
  - Tensorflow フレームワークで TFRecord フォーマットデータを扱うとき
  - MXNet フレームワークで RecordIO フォーマットデータを扱うとき
- Chainer および PyTorch には，上記のような入出力専用フォーマットが存在しないため，PIPE モードによる速度向上の恩恵を受けることはできない

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

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

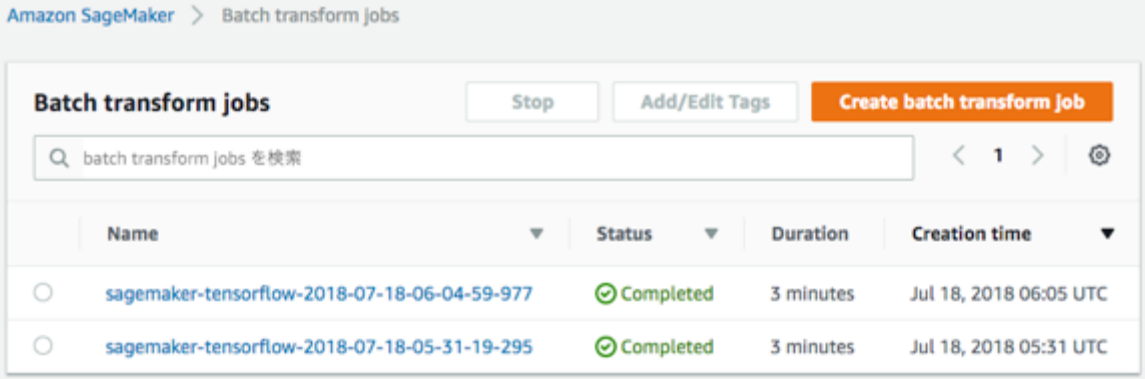
```
HTTP/1.1 200
Content-Type: ContentType
x-Amzn-Invoked-Production-Variant: InvokedProductionVariant

Body
```

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

aws

Transform Job によるバッチ推論

- 作成したモデルを指定して，バッチ推論を行うための Transform Job を作成する
  - バッチ処理を行うためのインスタンス数とインスタンスタイプを指定
  - バッチ推論が終了するとインスタンスも自動的に落とされる
- 推論対象のデータも，推論結果も S3 を使用



その他

SageMaker 以外での Chianer の利用について

- SageMaker を使えば，ChainerMN による分散学習が簡単に行える
- 諸事情により SageMaker ではなく，EC2 上で Chainer の分散学習を行いたい場合には，PFN 社より提供されている CloudFormation テンプレートを利用して，ChainerMN の実行環境を構築することが可能

https://research.preferred.jp/2018/06/chainermn-on-aws-with-cloudformation/

セキュリティ: 暗号化とコンプライアンス

- 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，およびエンドポイントインスタンスのストレージ
  - バッチ推論時の入出力データ
- Cloudtrail に対応済み
- PCI DSS および HIPPA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/
https://aws.amazon.com/about-aws/whats-new/2018/04/access-amazon-vpc-resources-for-training-and-hosting-with-amazon-sageMaker/
https://aws.amazon.com/about-aws/whats-new/2018/05/Amazon-SageMaker-Achieves-HIPAA-Eligibility/
https://aws.amazon.com/jp/about-aws/whats-new/2018/06/amazon-sagemaker-inference-calls-are-supported-on-aws-privatelink/

セキュリティ: 閉域網での通信

- SageMaker と S3 のデータ通信は，すべて S3 VPC エンドポイント経由で行うことが可能
  - 学習ジョブの入出力における S3 アクセス
  - 学習済モデルをデプロイする際の S3 アクセス
- SageMaker の API は，すべて PrivateLink 経由で行うことが可能
  - SageMaker Notebook Endpoint
  - SageMaker Service API
  - SageMaker Runtime API

https://aws.amazon.com/blogs/machine-learning/direct-access-to-amazon-sagemaker-notebooks-from-amazon-vpc-by-using-an-aws-privatelink-endpoint/
https://aws.amazon.com/about-aws/whats-new/2018/04/access-amazon-vpc-resources-for-training-and-hosting-with-amazon-sageMaker/
https://aws.amazon.com/jp/about-aws/whats-new/2018/06/amazon-sagemaker-inference-calls-are-supported-on-aws-privatelink/
https://aws.amazon.com/about-aws/whats-new/2018/08/amazon-sagemaker-apis-supported-on-aws-privatelink/

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
- Object Detection
- seq2seq
- K-means
- PCA
- LDA
- Neural Topic Model
- DeepAR Forecasting
- BlazingText (word2vec)
- Random Cut Forest
- K-NN

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 で分散学習と，ローカルモードでの学習を行う
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_distributed_mnist/tensorflow_distributed_mnist.ipynb
  - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_distributed_mnist/tensorflow_local_mode_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
- そのほかの Deep Learning フレームワーク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

SageMaker ハンズオン

SageMakerへのアクセス

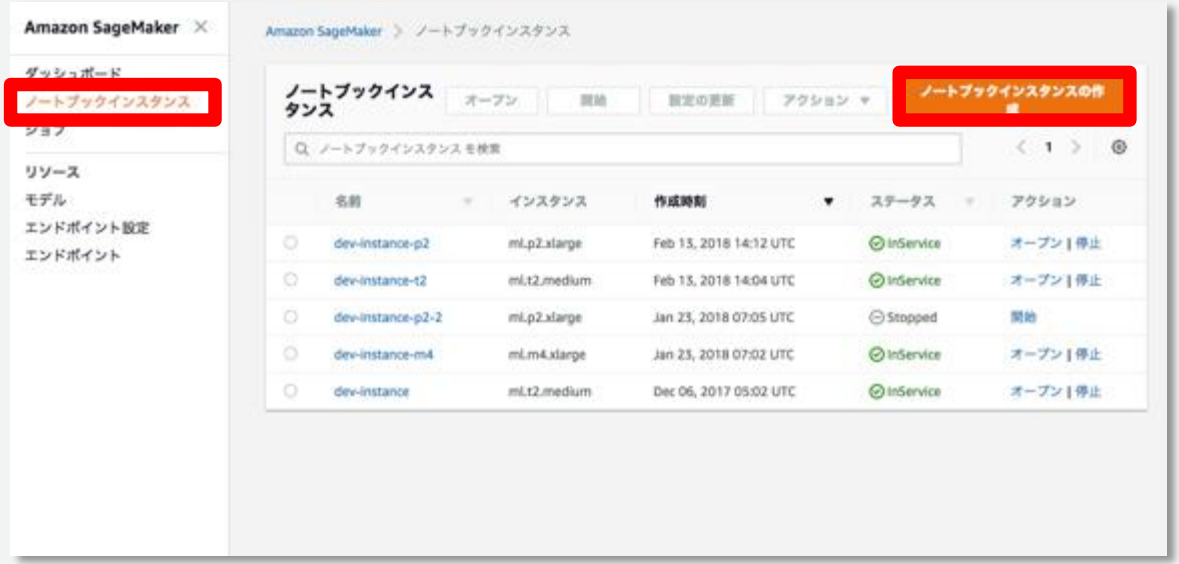
https://aws.amazon.com/jp/console/
(Chrome, Firefoxをご利用ください。IE, Safariは非対応です。)
マネジメントコンソール画面の検索ウィンドウに
SageMakerと入力して，サービスをクリックしてください。

リージョンの設定

ハンズオンでは**東京**リージョンを使用しますので、
画面右上から設定をお願いします。

ノートブックインスタンスの立ち上げまで

SageMaker コンソールを開き，左ペインのノートブックインスタンスを選択し，ノートブックインスタンスの作成をクリック



ノートブックインスタンス作成時の設定

- インスタンス名は自由に設定してください。
- ノートブックインスタンスは開発用に、安価な**ml.t2.medium**を選びます。
- IAM ロールは，**新しいロールの作成**を選びます。（作成方法は次で）

IAMロールの作成

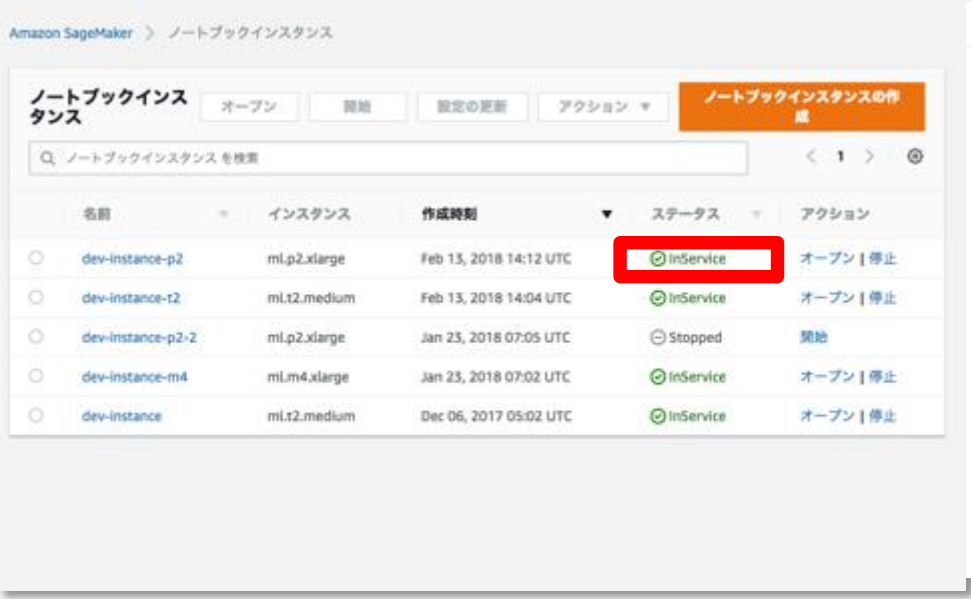
S3バケットのオプションを**任意のS3バケット**にしてロールを作成します。

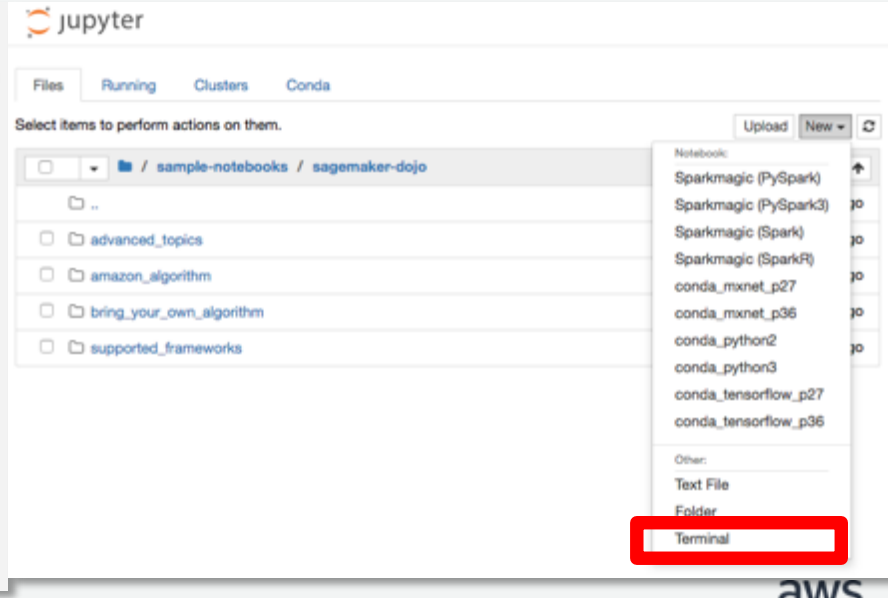
それ以外の設定

- VPC, ライフサイクル設定、暗号化はなし
- デフォルトのままノートブックインスタンスを作成してください。

Jupyter Notebook を開く

- ステータスが InService になったら，画面右側の「オープン」をクリックして，Jupyter Notebook を開く
- 画面右側の「New」→「Terminal」をクリックしてターミナルを開く





Terminal からノートブックのダウンロード

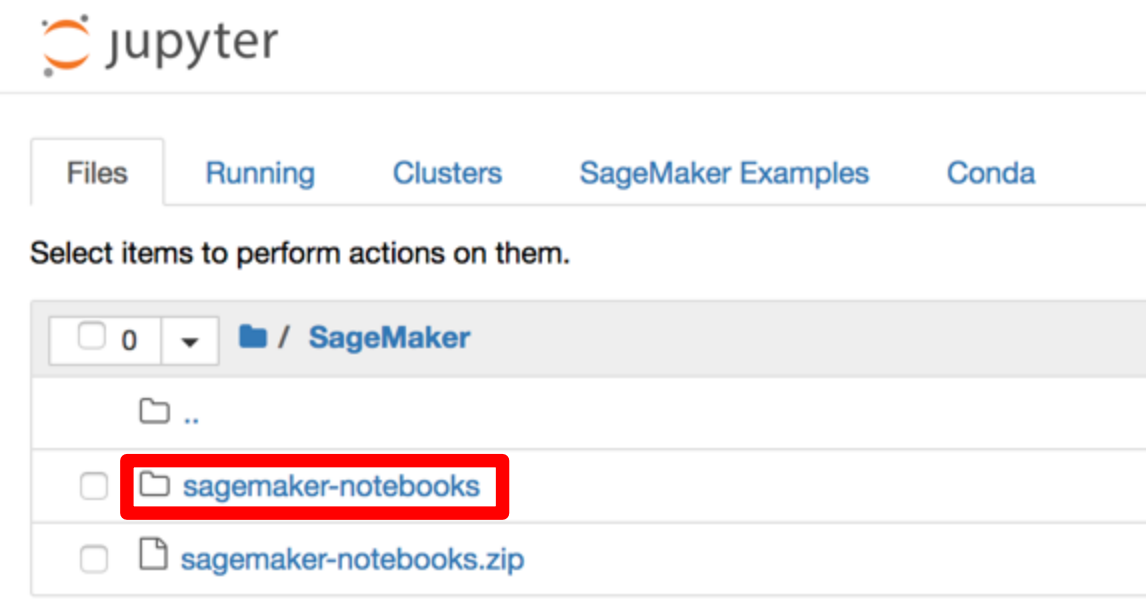
```
cd SageMaker
wget https://bit.ly/sagemaker-notebooks -O sagemaker-notebooks.zip
unzip sagemaker-notebooks.zip
```

大文字のO(オー)

```
sh-4.2$ cd SageMaker/
sh-4.2$ wget http://bit.ly/sagemaker-handson-notebooks
--2018-05-09 05:17:05--  http://bit.ly/sagemaker-handson-notebooks
Resolving bit.ly (bit.ly)... 67.199.248.10, 67.199.248.11
Connecting to bit.ly (bit.ly)|67.199.248.10|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://makotosh-tmp.s3.amazonaws.com/sagemaker-handson.zip?AWSAccessKeyId=AKIAI6INZRRPQFQNIWVQ&Expires=1605318423&Signatu
re=YoZI5VKtyt/KJG3YI2Xf08ijW4w%3D [following]
--2018-05-09 05:17:05--  https://makotosh-tmp.s3.amazonaws.com/sagemaker-handson.zip?AWSAccessKeyId=AKIAI6INZRRPQFQNIWVQ&Expires=160
5318423&Signature=YoZI5VKtyt/KJG3YI2Xf08ijW4w%3D
Resolving makotosh-tmp.s3.amazonaws.com (makotosh-tmp.s3.amazonaws.com)... 52.216.82.224
Connecting to makotosh-tmp.s3.amazonaws.com (makotosh-tmp.s3.amazonaws.com)|52.216.82.224|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 655115 (640K) [application/zip]
Saving to: 'sagemaker-handson-notebooks'

sagemaker-handson-notebooks      100%[=============================================================>] 639.76K  --.-KB/s    in 0.02s

2018-05-09 05:17:05 (27.0 MB/s) - 'sagemaker-handson-notebooks' saved [655115/655115]

sh-4.2$ unzip sagemaker-handson-notebooks
```

Jupyter Notebook に戻ってフォルダの確認

sagemaker-notebooksディレクトリがあり，中にノートブック群が含まれていることを確認



ハンズオンの内容

- ビルトインアルゴリズム
  - XGBoostを利用したMNISTの画像分類

- Chainer on SageMaker
  - ChainerでMLP（多層パーセプトロン）を実装し、MNIST画像分類

SageMakerですぐに使える組み込みアルゴリズム

機械学習アルゴリズム

アルゴリズム	説明	教師データ	用途
Linear Learner	線形回帰	あり	分類・回帰などの分析
XGBoost (eXtreme Gradient Boosting)	勾配ブーストツリーアルリズム	あり	分類・回帰などの分析
PCA	主成分分析法	なし	次元削減
k-means	K平均法	なし	クラスタリング
k-NN	K近傍法	あり	クラスタリング
Factorization Machines	分類・回帰	あり	レコメンド
Random Cut Forest (Amazon)	貯蔵サンプリング	なし	異常検知
LDA (Latent Dirichlet Allocation)	生成的統計モデル	あり	トピックモデル

※　LDAのオリジナルは教師なし

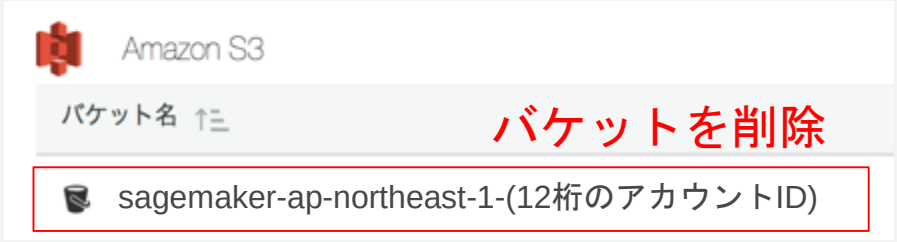
SageMakerですぐに使える組み込みアルゴリズム

ディープラーニング モデル

	アルゴリズム	説明	教師データ	用途
画像処理系	Image classification	画像分類	あり	画像の分類
	Object Detection	物体検出	あり	複数の画像を検出
	Semantic Segmentation	セグメンテーション	あり	画像中の物体の領域検出
自然言語系	seq2seq	Sockeye	あり	テキスト要約、音声認識
	Neural Topic Model	離散データのコレクションの潜在表現	なし	テキスト要約、音声認識
	Blazing text (Amazon)	Word2vec アルゴリズム実装	あり、なし	単語のマイニング
	Object2Vec	高次元分類	あり	レコメンド、文書埋め込み
時系列	DeepAR Forecasting	再帰型ニューラルネットワーク (RNN)	あり	時系列予測

今回使用したバケット・インスタンスの削除

- **今回使用したS3バケットの削除**



- **ノートブックインスタンスの削除**

- **エンドポイントの削除**

MLサービス最新Update
(Re:Inventでの発表分）

今回のアップデートの概要

1. **万人が対象**
   API サービス - 誰でも機械学習技術を活用できるように

2. **[機械学習] 技術者が対象**
   ML パイプライン - 機械学習のあらゆるプロセスを便利に

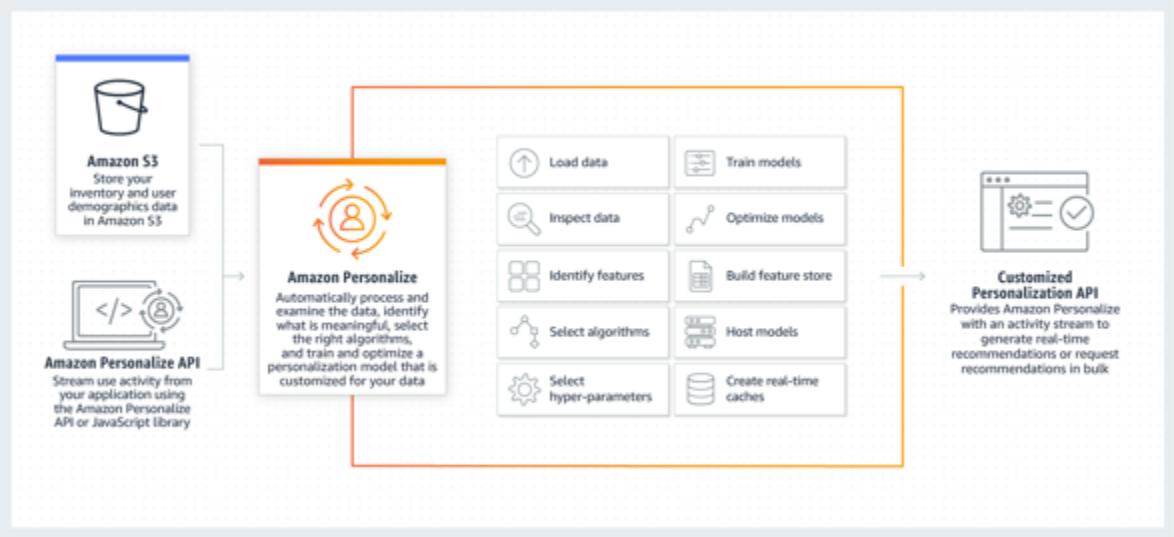
3. **機械学習技術者が対象**
   強化学習 – 機械学習の新たな領域

API サービス
誰でも機械学習技術を活用できるように

AWS re:Invent

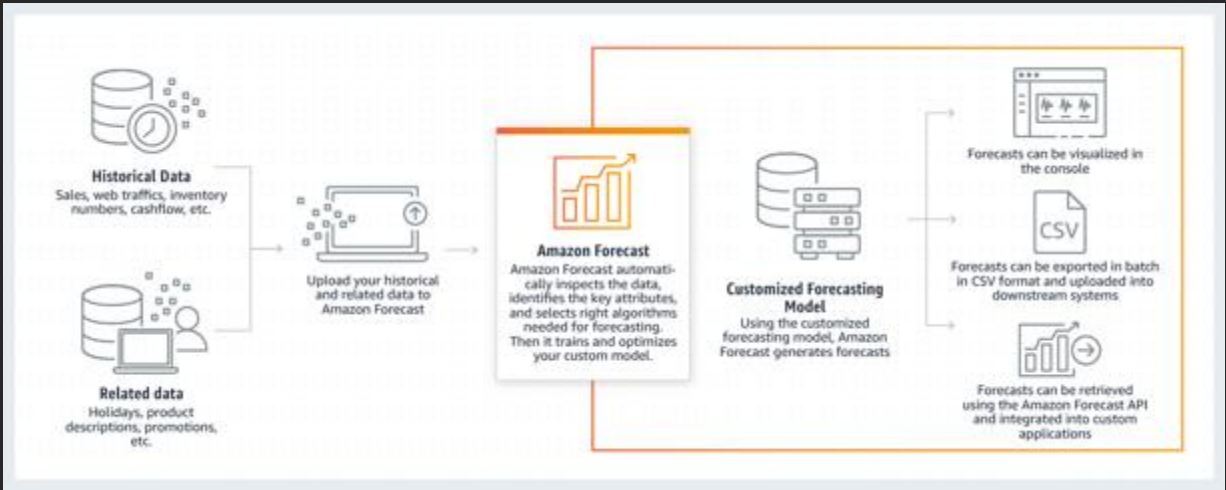
Amazon Personalize – easy-to-use なパーソナライズ

- Amazon Personalize は機械学習のエキスパートでない人向けの，レコメンデーションやパーソナライゼーションが簡単に行えるサービス
- Amazon が培ったレコメンデーションのナレッジを活用して，データを入れたら自動でレコメンド・パーソナライズの結果を取得可能
- 既存サービスに簡単にレコメンド機能を追加できるように



Amazon Forecast – easy-to-use な時系列予測サービス

- Amazon Forecast は機械学習のエキスパートでない人向けの，時系列データの予測を簡単に行えるサービス
- Amazon が培った時系列予測のナレッジを活用して，データを入れたら自動で時系列予測の結果を表示
- ウェブトラフィック，売上データ，商品需要，小売における製品値引きやプロモーションの効果などを予測する際に役立つ



Amazon Textract – 文章構造も認識可能な OCR

概要

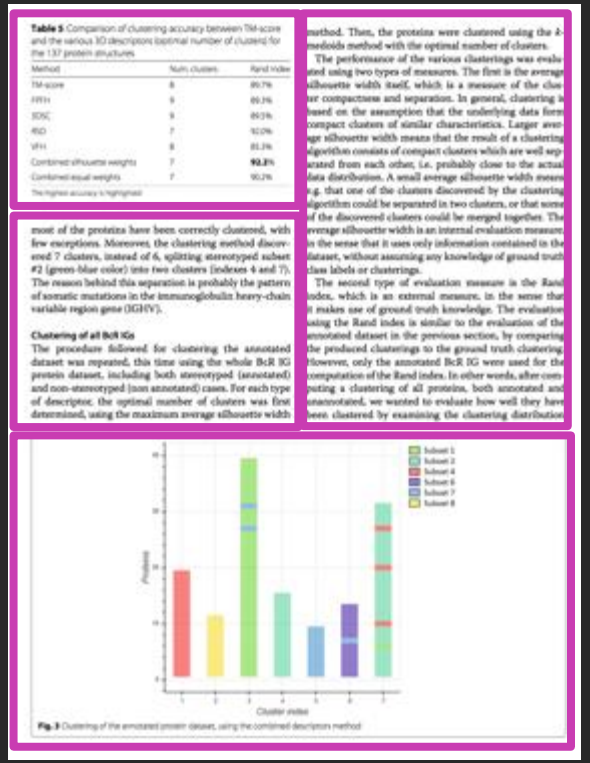
- Amazon Textract は高機能な OCR を提供
- 見出し，ヘッダー，フォーム，テーブル等の構造情報も認識
- 既存の OCR にあるような文書章フォーマットのテンプレートをメンテナンスする必要はない

価格

- $1.50 / 1000 pages

ステータス

- 以下のページからプレビュー申請可能
  https://pages.awscloud.com/textract-preview.html



Amazon Comprehend Medical – メディカルデータ対応

- 文章から様々な情報を抽出する，自然言語理解サービス Amazon Comprehend の拡張サービスで，メディカルデータ向けのチューニングを行なったもの
- 症状，治療法，薬等の医療用語，略語に対応できるように追加学習済み
- 医療上の文脈やエンティティの関係性を認識
- Protected health Information (PHI) を抜き出す API も用意．ただし 100 % の精度で検出できるわけではないので，その点に注意して使用する必要あり

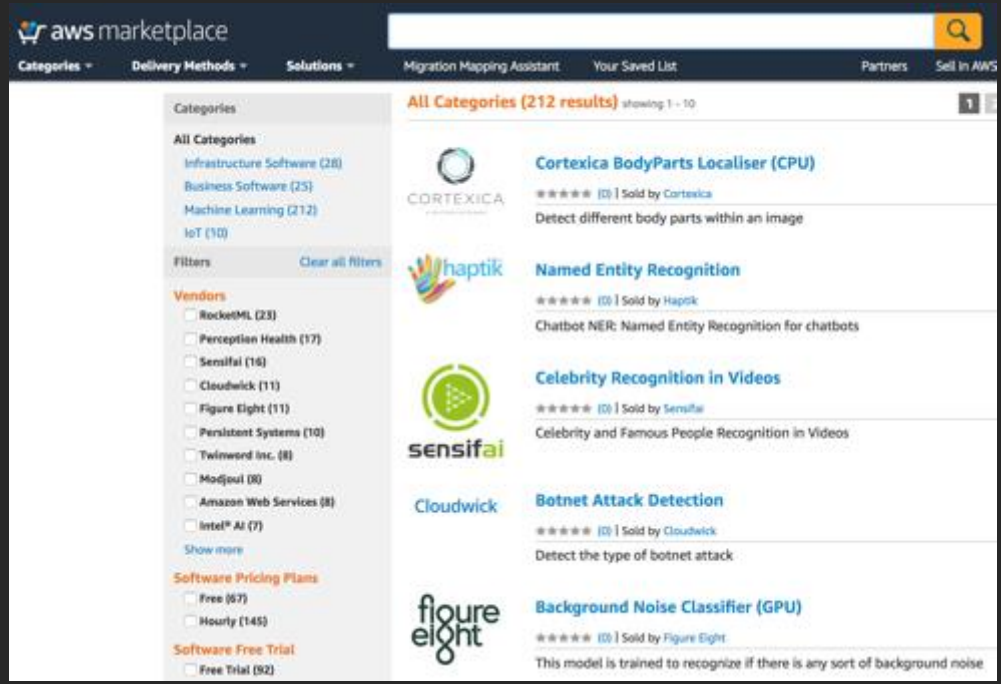
Amazon Translate Custom Terminology – カスタム語彙

- Amazon Translate は高機能な多言語間翻訳サービス
- ただ固有名詞等，決まった特定の形で処理をしたいものに関しては，必ずしもうまく対応できなかった
- カスタム語彙を追加することで，この問題に対応できるように
  - ただし多用しすぎると翻訳の質を逆に下げてしまう
  - 商品名，固有名詞のような常に一定の規則で変換されるもの留めることを推奨
- 言語ごとのカスタム語彙利用推奨，非推奨は右図の通り

言語	対応
東アジア言語 (中国語、日本語、韓国語、インドネシア語など)	推奨
ゲルマン語 (ドイツ語、オランダ語、英語、スウェーデン語、デンマーク語)	推奨
ロマンス語 (イタリア語、フランス語、スペイン語、ポルトガル語)	推奨
ヘブライ語	推奨
スラブ語 (ロシア語、ポーランド語、チェコ語)	非推奨
フィンウゴル語 (フィンランド語)	非推奨
アラビア語	非推奨
トルコ語	非推奨

ML Models in AWS Marketplace – 機械学習モデルのマーケットプレイスを新たに提供開始

- さまざまな会社が提供する機械学習モデルを，マーケットプレイス経由でサブスクライブし，Amazon SageMaker の学習ジョブおよび，推論エンドポイントやバッチ推論ジョブで利用可能に
- 200 以上のアルゴリズムがすでに公開済み
- 自社アルゴリズムの販売も当然可能



ML パイプライン
機械学習のあらゆるプロセスを便利に

AWS re:Invent

Amazon SageMaker Ground Truth –
効率的アノテーション作業の支援サービス

- 機械学習における教師データ（正解データ）の作成作業をサポートするマネージドサービス
- 高精度な推論を行うためには、正しいラベル付けがなされた教師データが必要だが、ラベル付けには多大な労力がかかるのでこれを簡単化するのが目的
- アノテーションを効率的に進めるためのツール提供のみならず、自社・他社双方のアノテーターを利用できる。Mechanical Turkとも連携可能
- 東京を含む5リージョンで利用可能。ラベル付けされたオブジェクト数に応じて課金となる

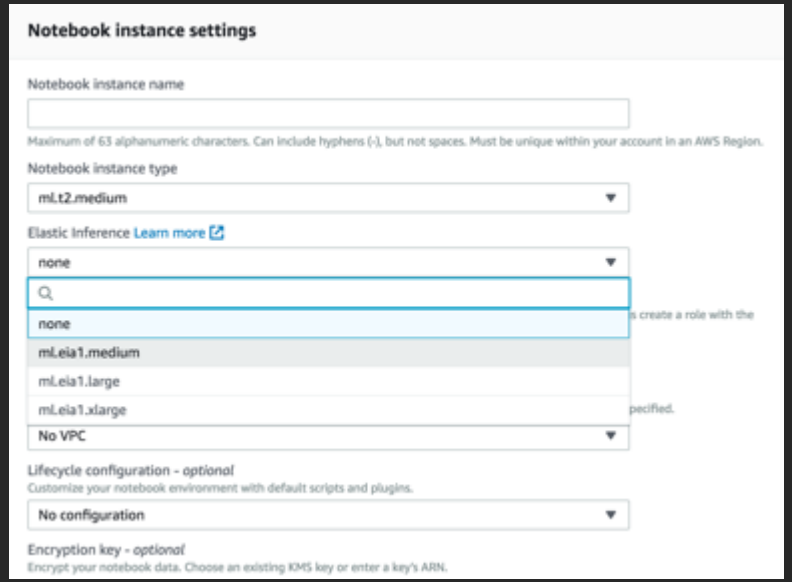
AWS re:Invent

Amazon SageMaker Neo –
各種フレームワーク対応のモデル変換

- Amazon SageMaker Neo は Tensorflow や PyTorch などの Deep Learning フレームワークで学習したモデルを，EC2 インスタンスや Greengrass デバイス上で高速に動作するように変換するサービス
- モデルの速度は最大 2 倍に（もちろん予測精度を一切下げることなく）
- 従来のフレームワーク上で 500MB-1GB あるようなモデルが，Amazon SageMaker Neo Runtime 上で 1MB 程度のサイズに
- Apache Software License で OSS として提供

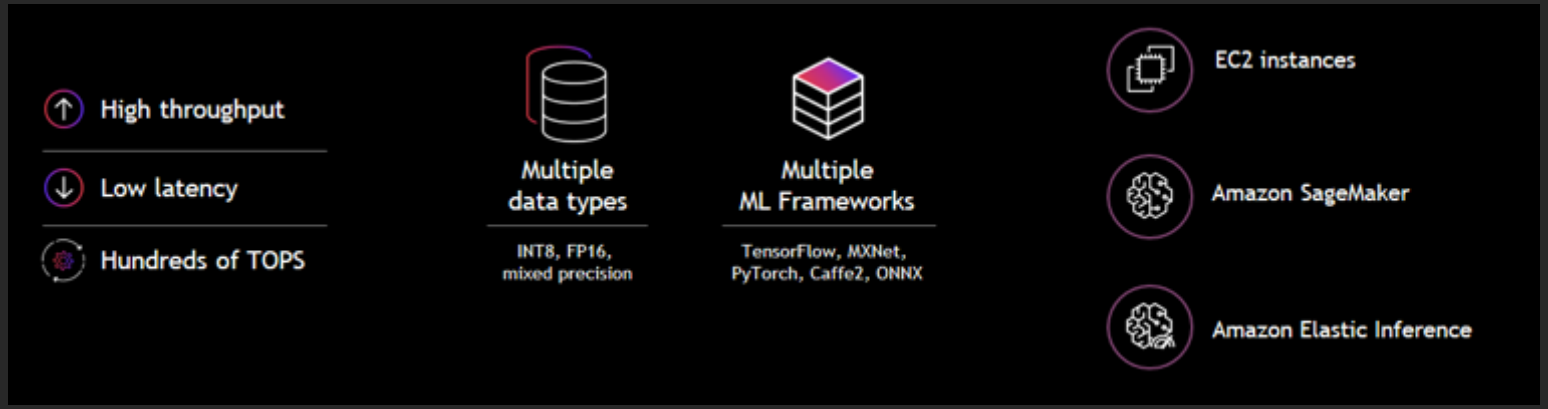
Amazon Elastic Inference –GPU による推論を安価に

- Amazon Elastic Inference は GPU リソースを推論演算ごとに細切れで利用可能とすることで，推論にかかる費用を削減しながら高速な推論を行えるサービス
- 現在の GPU は，主に学習プロセスに最適化されており，これを推論で用いる場合，コスト的な無駄が生じやすかった
- 推論に適した CPU / メモリの EC2 インスタンスを選んだ上で，GPU のスループットを得るための EIA (Elastic Inference Accelerator) を以下の 3 つから選択
  - eial.medium: 8TFLOPS の混合精度演算
  - eial.large: 16TFLOPS の混合精度演算
  - eial.xlarge: 32TFLOPS の混合精度演算



AWS Inferentia – 機械学習の推論チップ

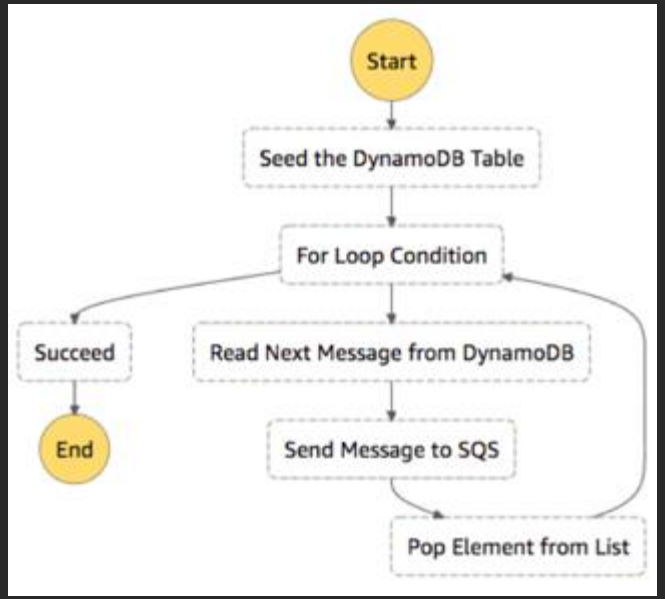
- AWS Inferentia は，低コストで高性能を実現するように設計された機械学習の推論チップ
- TensorFlow，Apache MXNet，PyTorch ONNX フォーマットを使用するモデルをサポートし，アプリケーションの計算コストの90%を節約することができる
- 2019 年に提供開始予定で，Amazon SageMaker，Amazon EC2，Amazon Elastic Inference が対応



AWS Step Functions API Connectors

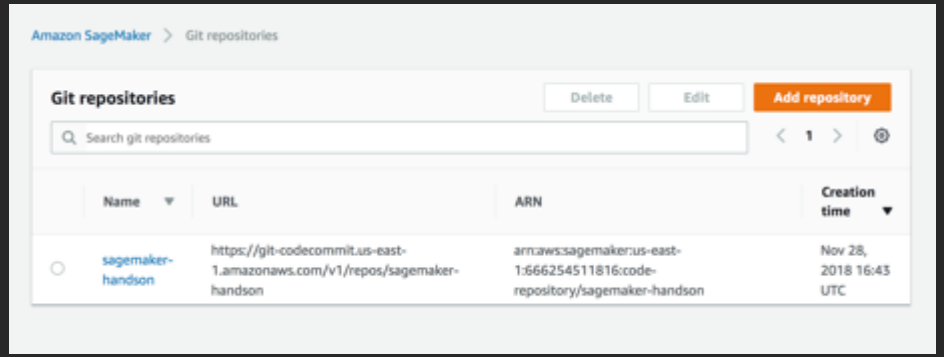
Step Functions のステートマシンから他のAWSサービス群に対して直接操作できるようになり，前処理～ジョブ実行～デプロイの流れをより便利に行えるように

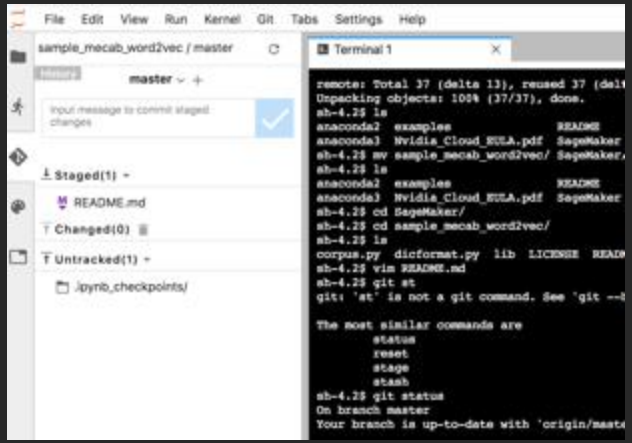
- DynamoDB: 既存のテーブルからitemの取り出し，新規itemの追加
- AWS Batch: バッチジョブの開始と完了待機
- Amazon ECS/Fargate: ECSまたはFargateのタスクを実行
- Amazon SNS: SNSトピックにメッセージをパブリッシュ
- Amazon SQS: キューにメッセージをプッシュ
- AWS Glue: ジョブを開始
- Amazon SageMaker: 学習ジョブ、変換ジョブを開始



Github / CodeCommit インテグレーション

- SageMaker のノートブックインスタンスが，Github および CodeCommit と統合された形で利用可能になった
- JupyterLab モードから GUI で利用可能
- Git リポジトリを紐づけてノートブックインスタンスを起動することで，最初から当該リポジトリが含まれた状態で作かを開始できる



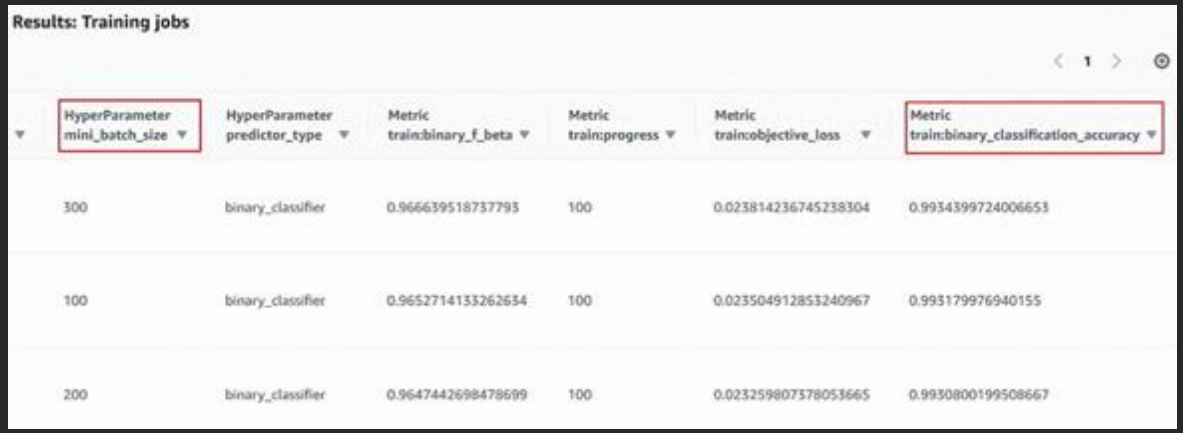


ビルトインアルゴリズムに
セマンティックセグメンテーションが追加

- セマンティックセグメンテーションは，ピクセル単位で画像の領域が何かを判定するもの
- Image Classification, Object Detection と並んでよく行われる画像認識の手法

学習ジョブに対して高度な検索を行える Search 機能（ベータ）が追加

- アルゴリズム，ハイパーパラメータ設定，学習データ，タグ等で，合致するデータを検索することが可能
- 検索結果を Accuracy や Loss 等のメトリクスでソートすることが可能
- デプロイされたモデルについて，どのデータが使われたかという Linage をトレースすることも可能



非常に高速な学習のための GPU インスタンス

- C5n インスタンス同様，最大 100Gbps のネットワーク性能
  - ENA ドライバが必要
  - 1 セッションでは 5Gbps が上限なので，通信真の多重化を考慮する必要あり
- 32GB 版 NVIDIA® Tesla® V100 GPU を 8 個， Intel® Xeon® Scalable (Skylake) の 96v CPU を搭載
- 2x900GB の NVMe SSD でディスクアクセスもさらに高速に

Model	GPUs	vCPU	Mem (GiB)	GPU Mem (GiB)	GPU P2P	Storage (GiB)	Dedicated EBS Bandwidth	Networking Performance
p3.2xlarge	1	8	61	16	-	EBS-Only	1.5 Gbps	Up to 10 Gigabit
p3.8xlarge	4	32	244	64	NVLink	EBS-Only	7 Gbps	10 Gigabit
p3.16xlarge	8	64	488	128	NVLink	EBS-Only	14 Gbps	25 Gigabit
p3dn.24xlarge*	8	96	768	256	NVLink	2 x 900 NVMe SSD	14 Gbps	100 Gigabit

強化学習
機械学習の新たな領域

AWS re:Invent

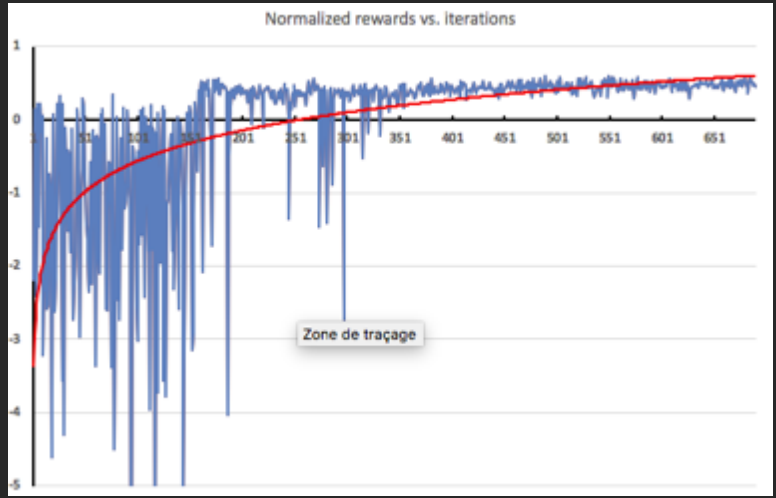
Amazon SageMaker で強化学習のサポート

- Amazon SageMaker RL は，強化学習を行うための機能拡張
- Open AI Gym / Intel Coach / Berkeley Ray RLLib などを含んだ形で，Tensorflow / MXNet のコンテナを利用することが可能
- また TensorForce や StableBaselines のような強化学習ライブラリを活用して，自分自身の環境を作成することも可能
- 以下のようなツール群と連携
  - シミュレーター
    - AWS が提供: AWS RoboMaker，Amazon Sumerian
    - 他社提供: MATLAB and Simulink（ライセンスは別途必要）
  - 環境: OpenAI Gym, Gym インタフェースを使った環境（Roboschool, EnergyPlus など）

SageMaker の強化学習でオートスケーリングの最適化

- **環境**: ロードプロファイルと稼働インスタンス数
- **行動**: インスタンスの起動 or 停止
- **報酬**: 費用 + トランザクション成功率.　キャパシティ不足の際は大きなペナルティ

```
rlestimator = RLEstimator(role=role,
          framework=Framework.TENSORFLOW,
          framework_version='1.11.0',
          toolkit=Toolkit.COACH,
          entry_point="train-autoscale.py",
          train_instance_count=1,
          train_instance_type=p3.2xlarge)
rlestimator.fit()
```



AWS DeepRacer – 強化学習のためのラジコンカー

- AWS DeepRacer は，1/18 スケールの 4 輪ラジコンカー
- 開発者が強化学習を始める際に，ハンズオンを行えるようにするためのもの

仕様

- 1/18 スケールラジコンカー
- Intel Atom プロセッサ
- 4M ピクセル, 1080p カメラ
- WiFi (802.11ac)
- 2h 以上稼働可能なバッテリー
- Ubuntu 16.04LTS, ROS (Robot Operating System), OpenVino

価格

- Amazon.com にて，$399.00 のところが今なら期間限定で $249.00 で予約受付中．2019/3/6 発売予定

まとめ

まとめ

1. **万人が対象** API サービス - 誰でも機械学習技術を活用できるように
   - 自分たちのデータで時系列予測・パーソナライズを簡単に行える新サービスが登場
   - 医療用自然言語解析，OCR といった，既存の言語・画像領域のサービスも拡大

2. **[機械学習] 技術者が対象** ML パイプライン - 機械学習のあらゆるプロセスを便利に
   - アノテーション，モデル変換，GPU 推論の最適化，ワークフロー管理など，ML パイプラインをエンドツーエンドでサポートするようなサービスを提供
   - 従来開発・の学習・推論プロセスに関しても，より多くの機能を提供

3. **機械学習技術者が対象** 強化学習 – 機械学習の新たな領域
   - Amazon SageMaker が強化学習をサポートし，他サービスとも連携
   - AWS DeepRacer によって，強化学習の第一歩を踏み出すことがより簡単に

aws