Amazon SageMaker Training (座学編)

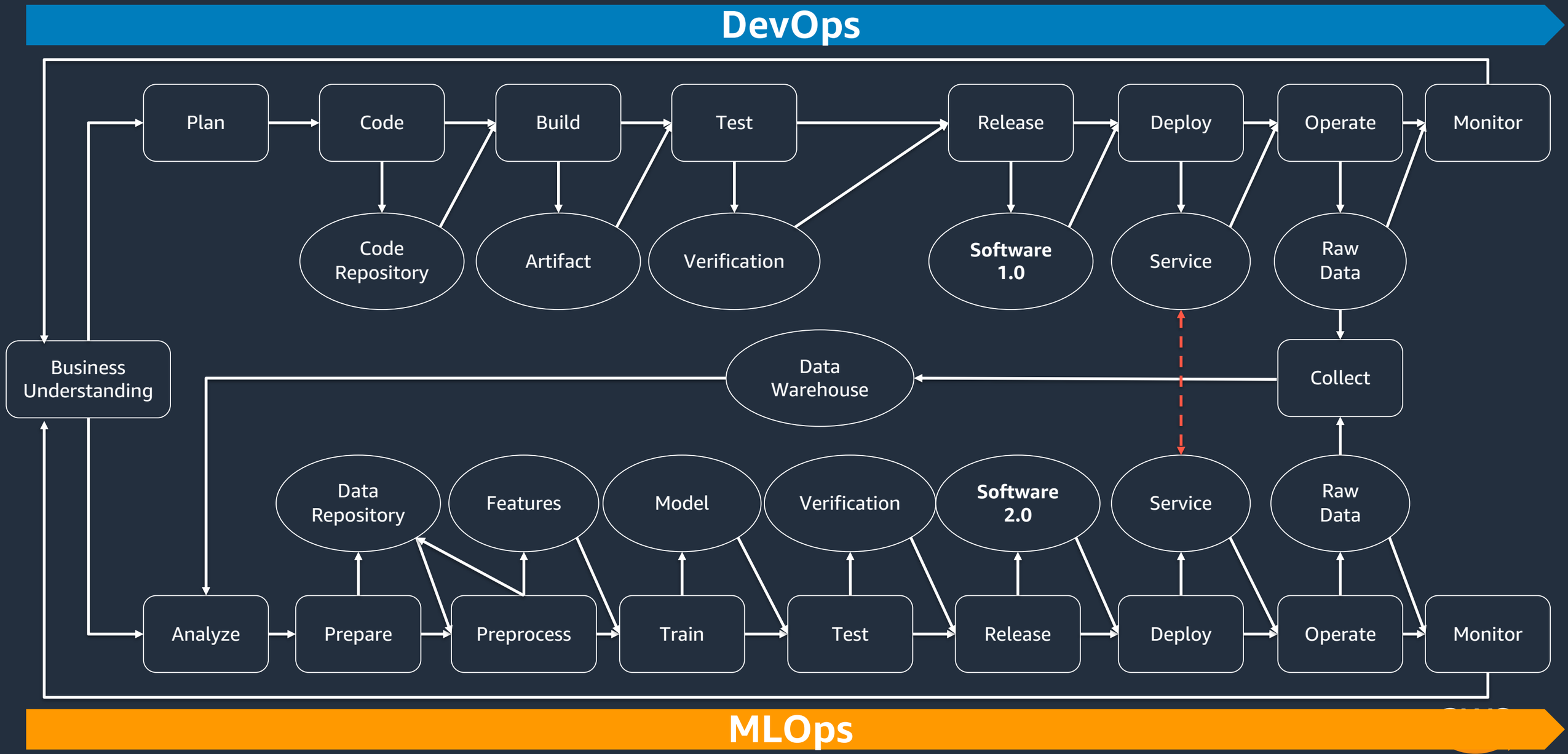
機械学習のモデル開発の試行錯誤を簡単にする

機械学習ソリューションアーキテクト
呉 和仁

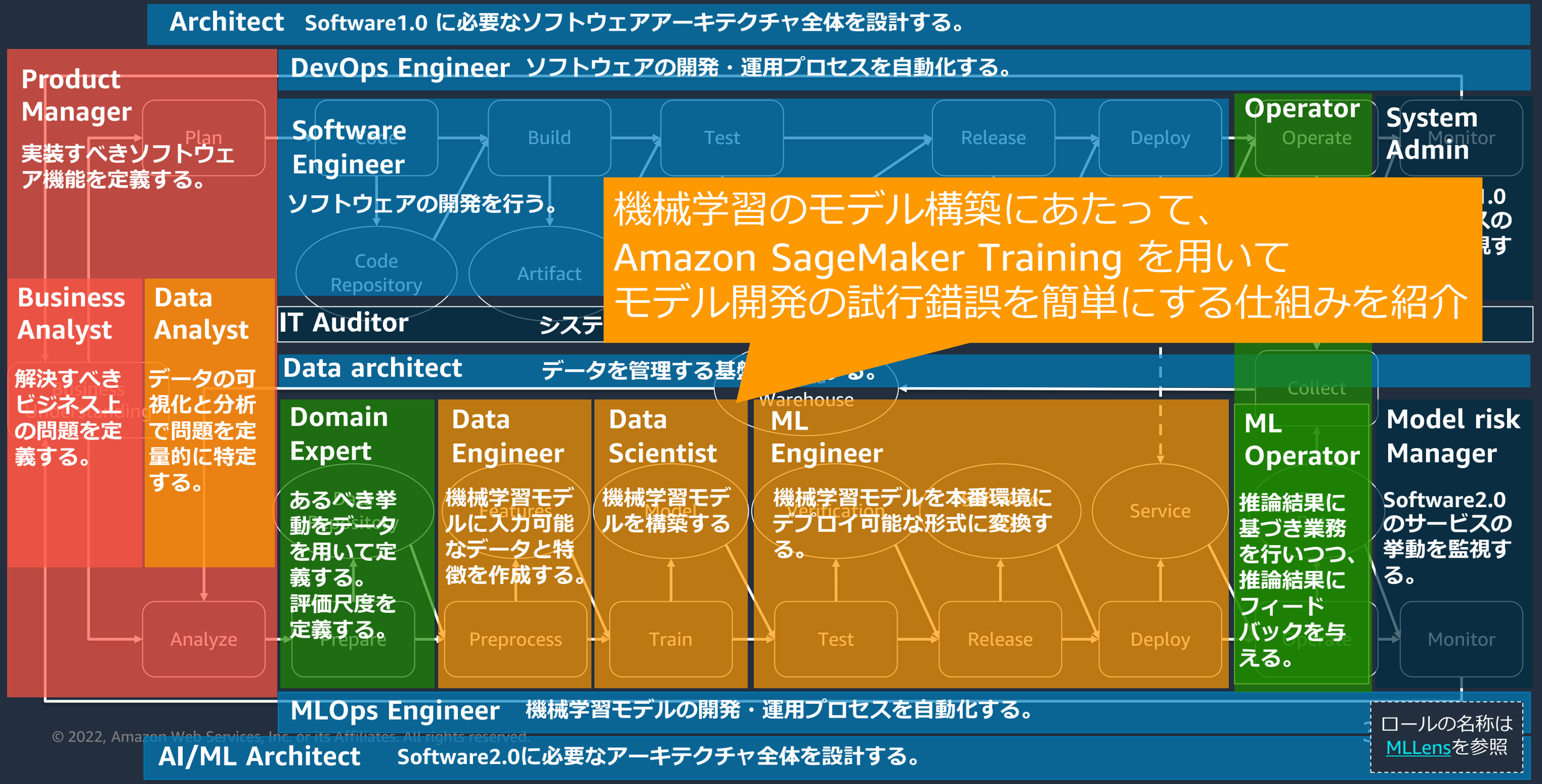
AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

DevOps & MLOps フロー



DevOps & MLOps を実現するロールマップ



この動画のゴール

- Amazon SageMaker Training の基本機能を理解して、
  セッション終了後すぐにコードを書いて実行できる

- 必要な事前知識
  - Python のコードの読み書きができること
  - 機械学習のチュートリアルを完了した程度の機械学習に関する知識

- またの機会に
  - ビルトインアルゴリズム
  - BYOC
  - 分散学習

  など

おしながき

- 機械学習のトレーニングで発生する課題
- Amazon SageMaker Training の概要
- 実際に動作するコードを用いたトレーニング開始方法
- Tips

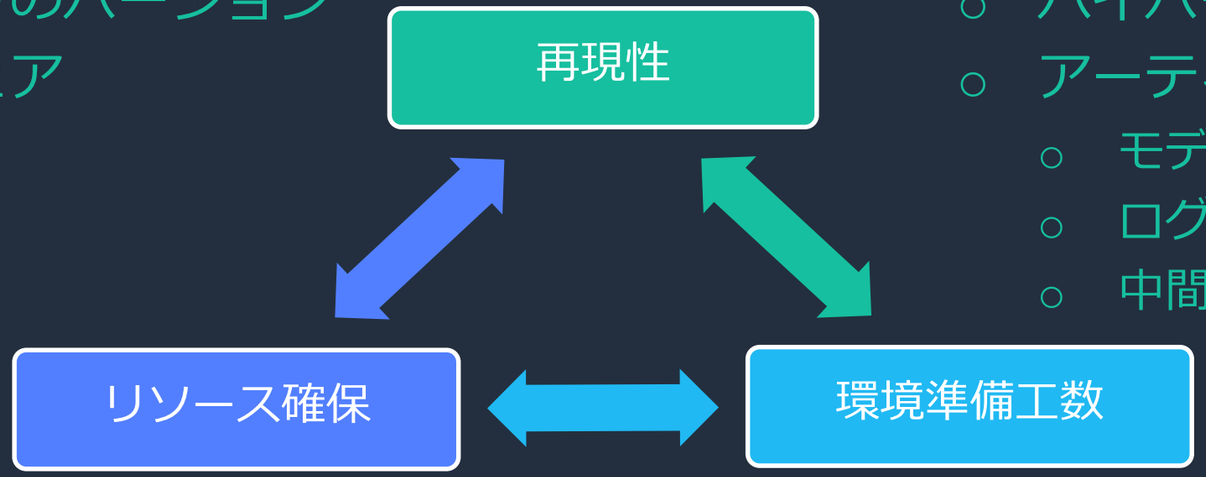
おしながき

- 機械学習のトレーニングで発生する課題
- Amazon SageMaker Training の概要
- 実際に動作するコードを用いたトレーニング開始方法
- Tips

機械学習のトレーニングでよくある課題

- 再現性とリソース確保と環境準備工数が密結合

- 実行環境
  - OS
  - ライブラリのバージョン
  - ハードウェア



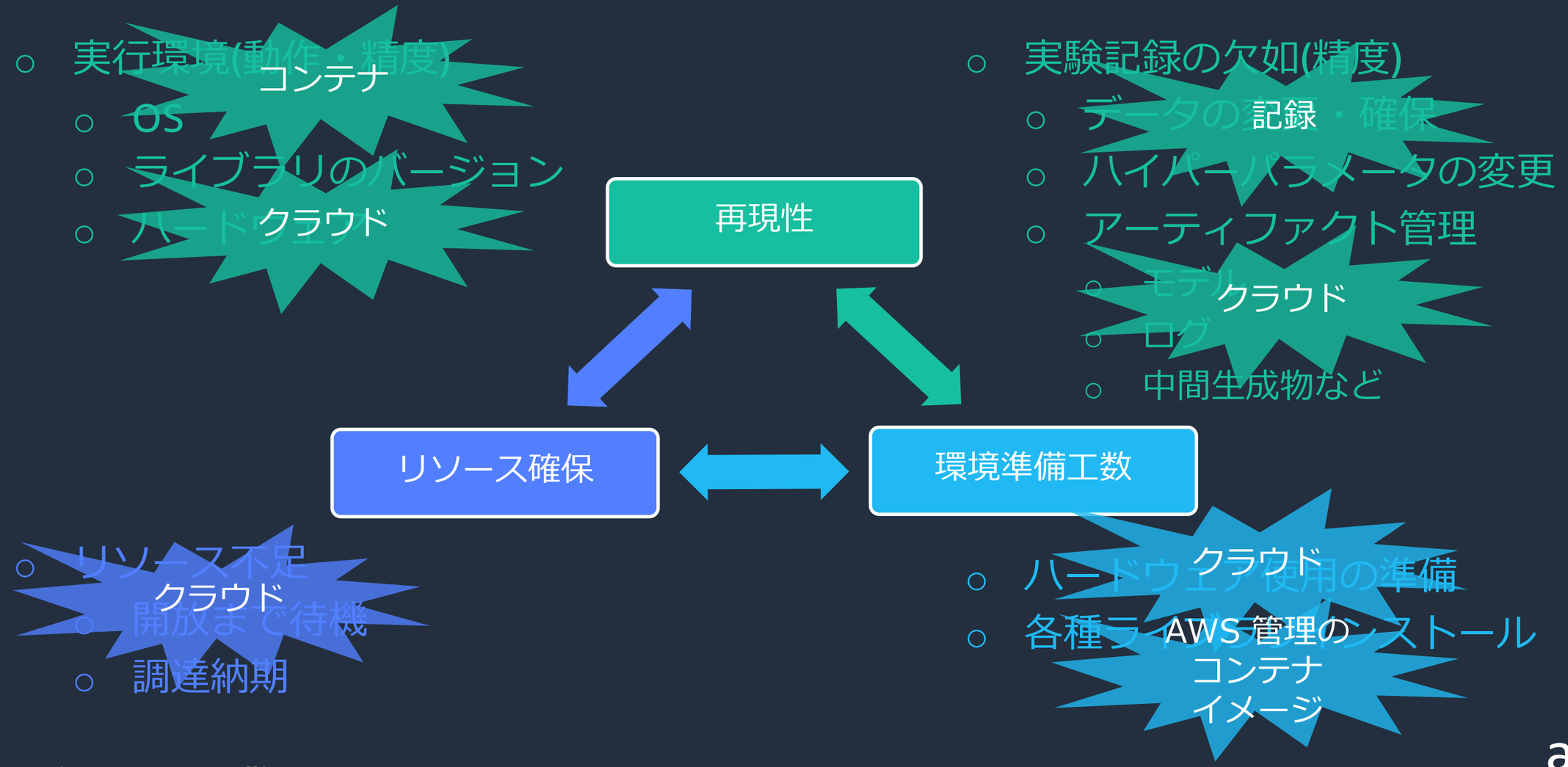
- 実験記録の欠如
  - データの変更・確保
  - ハイパーパラメータの変更
  - アーティファクト管理
    - モデル
    - ログ
    - 中間生成物など

- リソース不足
  - 開放まで待機
  - 調達納期

- ハードウェア使用の準備
- 各種ライブラリインストール

クラウドと周辺技術を適切に使えば解決できるものばかり

- 効率的にオーケストレーションする方法はないのか？



Amazon SageMaker Training でオーケストレーション

Amazon SageMaker Training が提供するサービスを 1 文 で表すならば

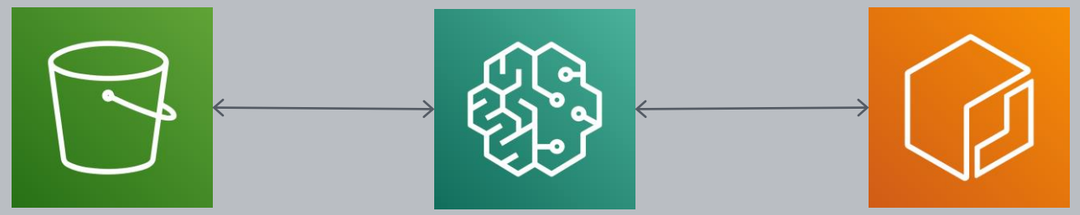
「用意したコード」と「用意したデータ」を
「指定したコンピューティングリソース」と「用意した環境」で実行し、
「実行履歴を自動記録」して「アーティファクトを自動で保存」する機能
を提供する

※ただし全てをユーザで用意する必要はなく AWS でも用意している

おしながき

- 機械学習のトレーニングで発生する課題
- Amazon SageMaker Training の概要
- 実際に動作するコードを用いたトレーニング開始方法
- Tips

Amazon SageMaker Training を構成する要素



Amazon S3　　Amazon SageMaker　　Amazon ECR

SageMaker
Python SDK

学習データ

学習スクリプト

DL / ML
実行環境

TensorFlow　PyTorch　mxnet　mxnet　mxnet　mxnet

aws

Amazon SageMaker Python SDK

- Amazon SageMaker Python SDKは、Amazon SageMaker 上で
  モデルの学習やデプロイなどを行うためのオープンソースライブラリ

- データサイエンティストが簡単かつ便利に AWS を使えるよう、
  Amazon SageMaker とその周辺のサービスに絞った高レベル API を提供

- pip でインストール可能で、手元の PC からも利用できる

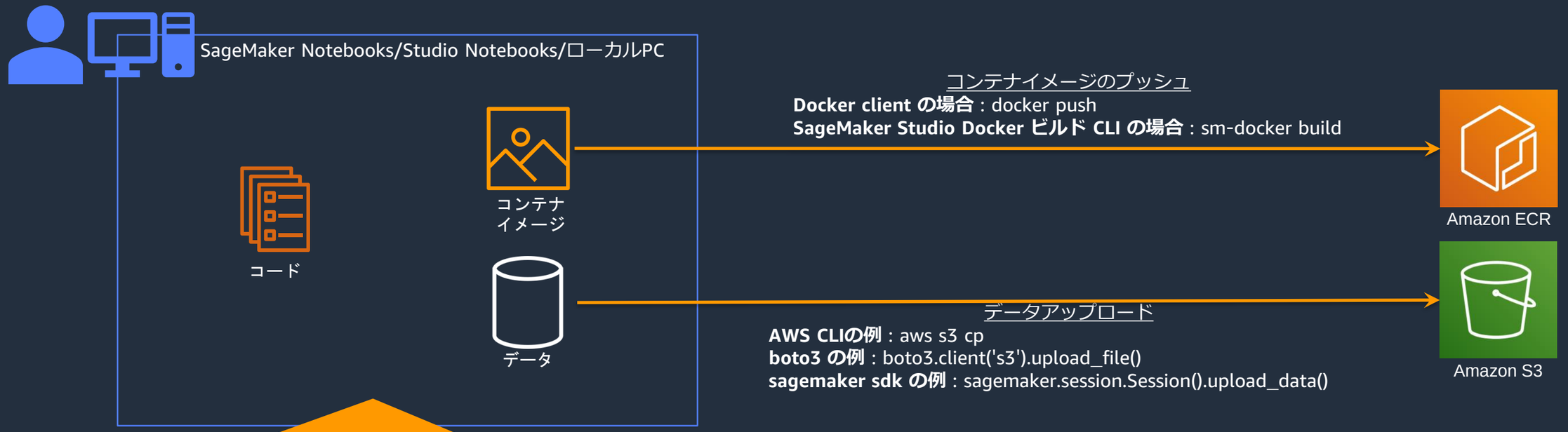
ソースコード： https://github.com/aws/sagemaker-python-sdk
Doc：https://sagemaker.readthedocs.io/en/stable/

※ Amazon SageMaker 自体は AWS CLI や boto3 などからも利用可能

SDK でトレーニングを開始する準備

（必要に応じて）

- トレーニングデータとトレーニング用のコンテナイメージを用意し、それぞれ Amazon S3 と Amazon ECR に格納する
- トレーニングコードを手元に用意する

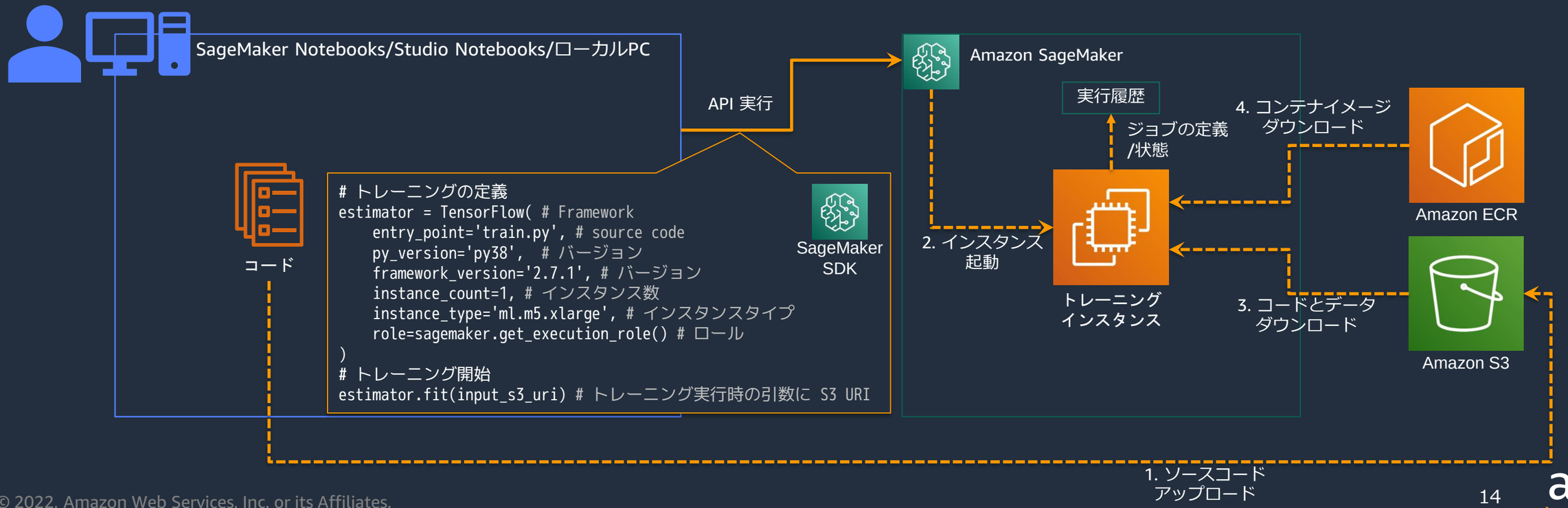


(ローカル PC の場合は事前に) pip install sagemaker
(sm-docker を使う場合は事前に) pip install sagemaker-studio-image-build (※)

※詳細はこちら
https://github.com/aws-samples/sagemaker-studio-image-build-cli

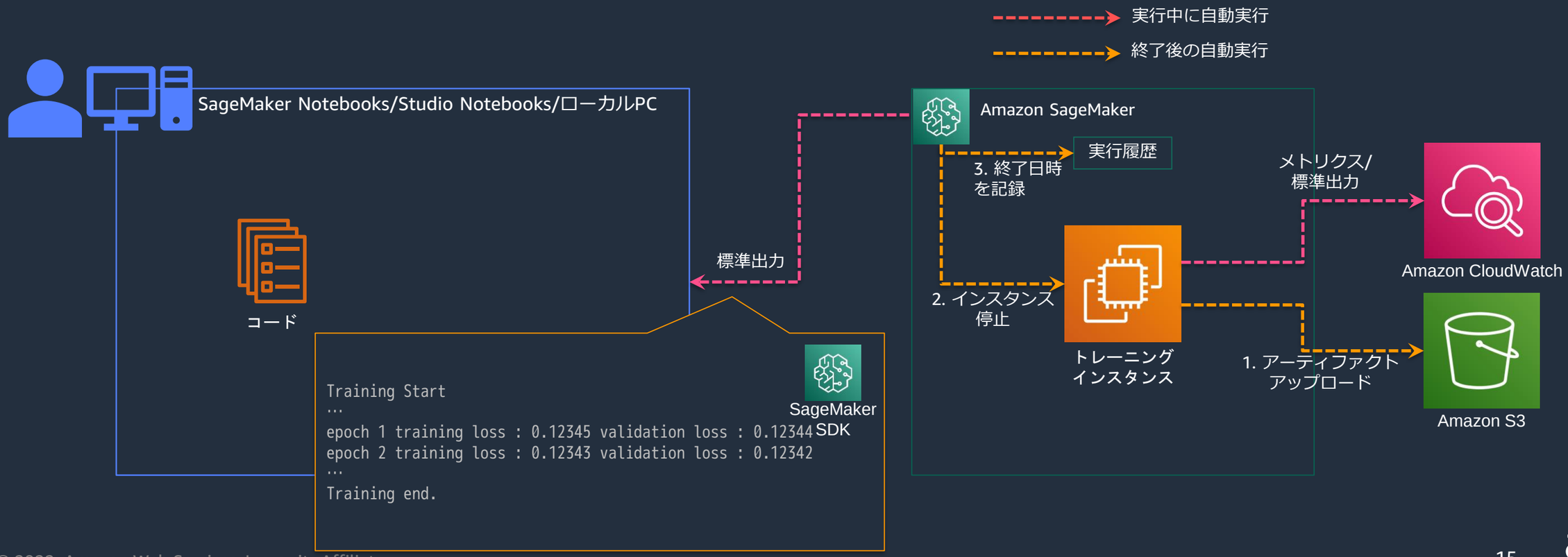
SDK の API でトレーニングを開始する

- トレーニングを定義して、トレーニングを開始
  - トレーニングコード(ローカル,もしくはGitリポジトリ), データ(S3 URI), コンテナ(イメージの URI, もしくはフレームワークとバージョンとインスタンスタイプ)を定義
  - 各フレームワークごとにジョブ定義用の Estimator クラスを利用して、使用するコードやバージョンを定義する



トレーニング完了時は自動でリソース削除

- 指定ディレクトリにアーティファクトを配置することで自動でトレーニング完了時にアーティファクトが Amazon S3 に転送
- ジョブ終了時に自動でインスタンスが削除され課金が停止
- 実行中の標準出力やメトリクスは CloudWatch に自動転送



SageMaker で使える AWS が管理・公開しているコンテナ

AWS が予め SageMaker に最適化された各フレームワークのコンテナイメージを準備しており、GPUや分散学習などを利用したトレーニングをすぐに開始可能

- AWS Deep Learning Container
  https://github.com/aws/deep-learning-containers
  - TensorFlow
  - PyTorch
  - MXNet
  - Hugging Face
- SageMaker Scikit-learn Container
  https://github.com/aws/sagemaker-scikit-learn-container
  - scikit-learn

おしながき

- 機械学習のトレーニングで発生する課題
- Amazon SageMaker Training の概要
- 実際に動作するコードを用いたトレーニング開始方法
- Tips

記載のコードは GitHub にて公開

- SageMaker Notebook や SageMaker Studio Notebook から以下のコマンドで取得可能
  git clone https://github.com/aws-samples/aws-ml-jp

- clone した後、下記ノートブックを参照してください。
  (ディレクトリ) sagemaker/sagemaker-traning/tutorial/
  (ノートブック) 1_hello_sagemaker_training.ipynb

- 詳細はデモ編の動画を参照してください。

1-1 用意したコードを実行する(トレーニングの実行)

各フレームワーク用の Estimator で entry_point 引数にコードを指定し、fit メソッドで実行

```
# 「用意したコード」./src/1-1/hello_sagemaker_training.py
print('Hello SageMaker Training')
exit()
```

```
# TensorFlow コンテナで Training Job
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-1/hello_sagemaker_training.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

```
# PyTorch コンテナで Training Job
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='./src/1-1/hello_sagemaker_training.py',
    py_version='py38',
    framework_version='1.9.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

```
# MXNet コンテナで Training Job
from sagemaker.mxnet import MXNet
estimator = MXNet(
    entry_point='./src/1-1/hello_sagemaker_training.py',
    py_version='py37',
    framework_version='1.8.0',
    instance_count=1,
    instance_type='ml.m4.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

```
# HuggingFace コンテナで  Training Job
from sagemaker.huggingface import HuggingFace
estimator = HuggingFace(
    entry_point='./src/1-1/hello_sagemaker_training.py',
    py_version='py37',
    transformers_version='4.6.1',
    tensorflow_version='2.4.1',
    instance_count=1,
    instance_type='ml.g4dn.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

```
# scikit-learn コンテナで Training Job
from sagemaker.sklearn import SKLearn
estimator = SKLearn(
    entry_point='./src/1-1/hello_sagemaker_training.py',
    py_version='py3',
    framework_version='0.23-1',
    instance_count=1,
    instance_type='ml.c5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

実行結果
(略)
Hello SageMaker Training
(略)

1-2-1 トレーニングインスタンスにデータを持ち込む 単一ファイル編(トレーニングデータを持ち込む)

Estimator の使い方は変わらず、fit メソッド実行時に持ち込みたいデータの URI を指定する。
upload_data は該当ファイルの S3 URI を返す。

```
# 「用意したデータ」./data/1-2-1/calc.txt
3+4
4-2
5*1
6/2
```

```
# 「用意したコード」./src/1-2-1/calc.py
import os
input_dir = os.environ.get('SM_CHANNEL_TRAINING')
input_txt_path = os.path.join(input_dir,os.listdir(input_dir)[0])
with open(input_txt_path,'rt') as f:
    input_text_lines = f.read()
for input_text_line in input_text_lines.split('¥n'):
    print(eval(input_text_line))
exit()
```

```
# S3 にデータをアップロード
import sagemaker
input_s3_uri = sagemaker.session.Session().upload_data(
    path='./data/1-2-1/calc.txt',
    bucket=sagemaker.session.Session().default_bucket(),
    key_prefix='training/1-2'
)
```

```
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-2-1/calc.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit(input_s3_uri)
```

コード実行前に当該データが、トレーニングインスタンスの
環境変数 SM_CHANNEL_TRAINING(=/opt/ml/input/data/training/)
指定されるディレクトリに転送される

実行結果

```
(略)
7
2
5
3.0
(略)
```

1-2-2 トレーニングインスタンスにデータを持ち込む 単一ディレクトリ編(トレーニングデータを持ち込む)

アップロードする際の指定を、ファイルからディレクトリにするとディレクトリ以下のファイル全てが転送される(upload_dataメソッドの返り値はファイルの URI から prefix になる)

```
# 「用意したデータ」./data/1-2-2/calc1.txt
2+2
3-1
```

```
# 「用意したデータ」./data/1-2-2/calc2.txt
5*1
16/2
```

```
# 「用意したコード」./src/1-2-2/calc.py
import os
input_dir = os.environ.get('SM_CHANNEL_TRAINING')
for file_name in sorted(os.listdir(input_dir)):
    input_txt_path = os.path.join(input_dir,file_name)
    with open(input_txt_path,'rt') as f:
        input_text_lines = f.read()
    for input_text_line in input_text_lines.split('¥n'):
        print(eval(input_text_line))
exit()
```

```
# S3 にデータをアップロード
import sagemaker
input_s3_uri = sagemaker.session.Session().upload_data(
    path='./data/1-2-2/',
    bucket=sagemaker.session.Session().default_bucket(),
    key_prefix='training/1-2-2'
)
```

```
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-2-2/calc.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit(input_s3_uri)
```

実行結果

```
(略)
4
2
5
8.0
(略)
```

ディレクトリ単位でアップロードしても
環境変数 SM_CHANNEL_TRAINING(=/opt/ml/input/data/training/)
指定されるディレクトリに転送されるのは変わらない

1-2-3 トレーニングインスタンスにデータを持ち込む 複数ディレクトリ編(トレーニングデータを持ち込む)

train用,validation用,test用といった分割や、交差検証のようにデータを分割した場合も
それぞれアップロードして、dict 型で渡すだけ

```
# 「用意したデータ」./data/1-2-3/fold1/calc.txt
1+2
3+4
```

```
# 「用意したデータ」./data/1-2-3/fold2/calc2.txt
5+6
7+8
```

```
# 「用意したコード」./src/1-2-3/calc.py
import os, json
channels = os.environ.get('SM_CHANNELS')
channels_list = json.loads(channels)
for channel in channels_list:
    input_dir = os.environ.get('SM_CHANNEL_' + channel.upper())
    for file_name in os.listdir(input_dir):
        input_txt_path = os.path.join(input_dir,file_name)
        with open(input_txt_path,'rt') as f:
            input_text_lines = f.read()
        for input_text_line in input_text_lines.split('¥n'):
            print(eval(input_text_line))
exit()
```

```
# S3 にデータをアップロード
import sagemaker
fold1_input_s3_uri = sagemaker.session.Session().upload_data(
    path='./data/1-2-3/fold1/', bucket=sagemaker.session.Session().default_bucket(),
    key_prefix='training/1-2-3/fold1'
)
fold2_input_s3_uri = sagemaker.session.Session().upload_data(
    path='./data/1-2-3/fold2/', bucket=sagemaker.session.Session().default_bucket(),
    key_prefix='training/1-2-3/fold2'
)
```

```
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-2-3/calc.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit({
    'fold1':fold1_input_s3_uri,
    'fold2':fold2_input_s3_uri,
})
```

実行結果

```
(略)
3
7
11
15
(略)
```

fitにdict型を入れることで、複数の S3 prefix 以下のファイルを転送できる。
転送先のリストは環境変数 SM_CHANNELS で保有しているチャネルリストから参照できる、
環境変数 SM_CHANNEL_{dictのキー} の値。この場合は、
SM_CHANNEL_FOLD1(=/opt/ml/input/data/fold1)と、
SM_CHANNEL_FOLD2(=/opt/ml/input/data/fold2)

1-3 アーティファクトを S3 に転送する(モデルの保存)

S3 に転送したいファイルを環境変数 SM_MODEL_DIR で指定されるディレクトリもしくは、環境変数 SM_OUTPUT_DATA_DIR で指定されるディレクトリに保存しておくと、ジョブ完了時に自動で S3 に転送される。

```
# 「用意したデータ」./data/1-3/data1.txt
1,2
3,4
```

```
# 「用意したデータ」./data/1-3/data2.txt
5,6
7,8
```

```
# 「用意したコード」./src/1-3/concat.py
import os, json
channels = os.environ.get('SM_CHANNELS')
model_dir = os.environ.get('SM_MODEL_DIR')
channels_list = json.loads(channels)
output_txt = ''
for channel in channels_list:
    input_dir = os.environ.get('SM_CHANNEL_' + channel.upper())
    for file_name in sorted(os.listdir(input_dir)):
        input_txt_path = os.path.join(input_dir,file_name)
        with open(input_txt_path,'rt') as f:
            output_txt += f.read()+'¥n'
with open(os.path.join(model_dir,'output.csv'),'wt') as f:
    f.write(output_txt)
print('processing completed')
exit()
```

```
# S3 にデータをアップロード
import sagemaker
input_s3_uri = sagemaker.session.Session().upload_data(
    path='./data/1-3/',
    bucket=sagemaker.session.Session().default_bucket(),
    key_prefix='training/1-3'
)
```

```
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-3/concat.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit(input_s3_uri)
```

実行結果

```
(略)
processing completed
(略)
```

SM_MODEL_DIR(=/opt/ml/model) に保存した場合は model.tar.gz
SM_OUTPUT_DATA_DIR(=/opt/ml/output/data) に保存した場合は output.tar.gz
というファイル名で S3 に転送。
SM_MODEL_DIR は SageMaker SDK でモデルをホスティングするときのモデルとしてそのまま使える機能を有する

アーティファクトの確認

- ジョブの実行結果にある S3ModelArtifacts から model.tar.gz の URI を抽出できる

```
import tarfile
# S3 URI を取得して prefix 部分を抽出
prefix = estimator.latest_training_job.describe()['ModelArtifacts']['S3ModelArtifacts'].replace(
    f's3://{sagemaker.session.Session().default_bucket()}/',''
)
# アーティファクトをダウンロード
sagemaker.session.Session().download_data(
    './', sagemaker.session.Session().default_bucket(), key_prefix=prefix
)
# 解凍
with tarfile.open('./model.tar.gz', 'r') as f:
    f.extractall()
# 中身を出力
with open('./output.csv','rt') as f:
    print(f.read())
```

実行結果

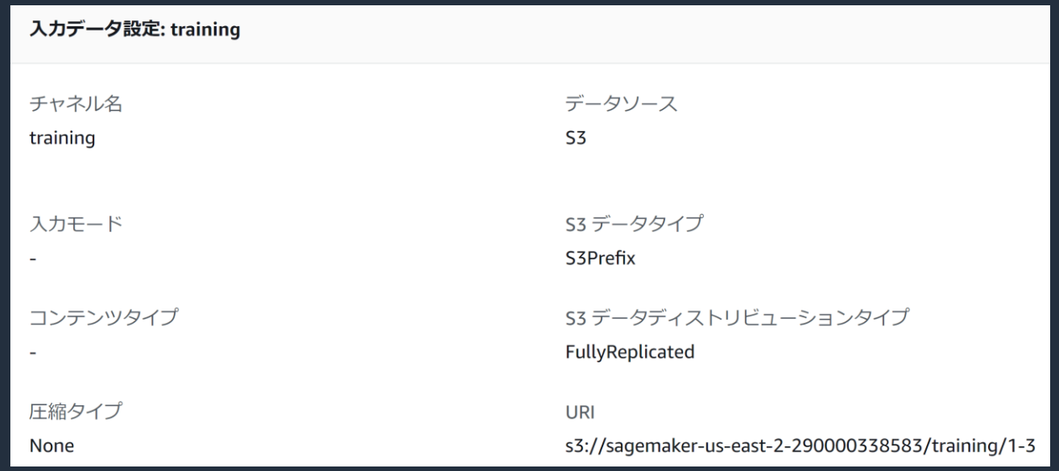
```
1,2
3,4
5,6
7,8
```

自動で記録されるものを確認

SageMaker SDK や boto3, マネジメントコンソールなどから確認できる

- コンテナイメージ
- 入力データ
- ソースコード
- 実行時間
- インスタンスタイプ
- アーティファクト
- ハイパーパラメータ(後述)

など




```
# 確認
estimator.latest_training_job.describe()
# 結果抜粋
{'TrainingJobName': 'training-job/tensorflow-training-YYYY-MM-DD-HH-MI-SS-mmm',
 'TrainingJobArn': 'arn:aws:sagemaker:{REGION}:{ACCOUNT ID}:training-job/tensorflow-training-YYYY-MM-DD-HH-MI-SS-mmm',
 'ModelArtifacts': {'S3ModelArtifacts': 's3://sagemaker-{REGION}-{ACCOUNT_ID}/tensorflow-training-YYYY-MM-DD-HH-MI-SS-mmm/output/model.tar.gz'},
 'sagemaker_program': '"concat.py"',
 'sagemaker_submit_directory': '"s3://sagemaker-{REGION}-{ACCOUNT_ID}/tensorflow-training-YYYY-MM-DD-HH-MI-SS-mmm/source/sourcedir.tar.gz"'},
 'AlgorithmSpecification': {'TrainingImage': '763104351884.dkr.ecr.us-east-2.amazonaws.com/tensorflow-training:2.7.1-cpu-py38'},
 'InputDataConfig': [
   {'ChannelName': 'training','DataSource': {'S3DataSource': {'S3DataType': 'S3Prefix','S3Uri': 's3://sagemaker-{REGION}-{ACCOUNT_ID}/training/1-3'}},}
 ],
 'ResourceConfig': {'InstanceType': 'ml.m5.xlarge','InstanceCount': 1,'VolumeSizeInGB': 30},
 'TrainingTimeInSeconds': 97,'BillableTimeInSeconds': 97,}
```

1-4 ライブラリや自作モジュールの追加

- コンテナイメージにないライブラリ等を使う場合は、コードのディレクトリ内に requirements.txt を入れるか、ソースコードを格納し、source_dir でコード一式を格納したディレクトリを指定する

```
# 「用意したコード」./src/1-4/requirements.txt
beautifulsoup4==4.9.3
```

```
# 「用意したコード」./src/1-4/my_module.py
def version_check(module):
    print(f' {module.__name__} version is {module.__version__}')
    return None
```

```
# 「用意したコード」./src/1-4/bs4_version_check.py
import bs4, my_module
my_module.version_check(bs4)
exit()
```

```
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='bs4_version_check.py',
    source_dir='./src/1-4',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role()
)
estimator.fit()
```

source_dir 引数を使い、そのルートディレクトリ(※)に requirements.txt を配置して必要なライブラリを記載することで、コード実行前に pip install −r requrirements.txt を自動で走らせることができる。
自作モジュールも source_dir で指定したディレクトリに配置することで使用可能。

(※実行時のカレントディレクトリは source_dir で指定したディレクトリー式を転送した /opt/ml/code で、Python 実行時のコマンドライン引数に entry_point で指定したファイル名が設定されるため)

実行結果

```
(略)
/usr/local/bin/python3.8 -m pip install -r requirements.txt
Collecting beautifulsoup4==4.9.3
Downloading beautifulsoup4-4.9.3-py3-none-any.whl (115 kB)
Collecting soupsieve>1.2
Downloading soupsieve-2.3.1-py3-none-any.whl (37 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.9.3 soupsieve-2.3.1
(略)
/usr/local/bin/python3.8 bs4_version_check.py --model_dir s3://sagemaker-
{REGION}-{ACCOUNT_ID}/tensorflow-training-YYYY-MM-DD-HH-MI-SS-mmm/model
bs4 version is 4.9.3.
(略)
```

1-5 ハイパーパラメータをコードの外部から入力

- Estimator インスタンス生成時の hyperparameters 引数で dict 型で入力すると環境変数及びコマンドライン引数に設定されるので、引き受けられるようにトレーニングコードを修正する

```python
# 「用意したコード」./src/1-5/hp_calc.py
import os, argparse
parser = argparse.ArgumentParser()
parser.add_argument('--first-num', type=int)
parser.add_argument('--second-num', type=int)
parser.add_argument('--operator', type=str)
parser.add_argument('--model_dir', type=str)
args = parser.parse_args()
if args.operator == 'p':
    print(f'The answer is {args.first_num + args.second_num}')
elif args.operator == 'm':
    print(f'The answer is {args.first_num - args.second_num}')
exit()
```

```python
# トレーニングジョブを実行
from sagemaker.tensorflow import TensorFlow
estimator = TensorFlow(
    entry_point='./src/1-5/hp_calc.py',
    py_version='py38',
    framework_version='2.7.1',
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role(),
    hyperparameters={
        'first-num':5,
        'second-num':2,
        'operator':'m'
    }
)
estimator.fit()
```

hyperparameters引数に指定した内容は、SM_HPS という環境変数に
json 形式で格納されるので
hps = json.loads(os.environ.get('SM_HPS'))
のように受けて辞書形式で使うか、
この例のように argparse を用いてパースして使う方法がある
※ いずれの方法でも default 値は設定したほうが使いやすい

実行結果

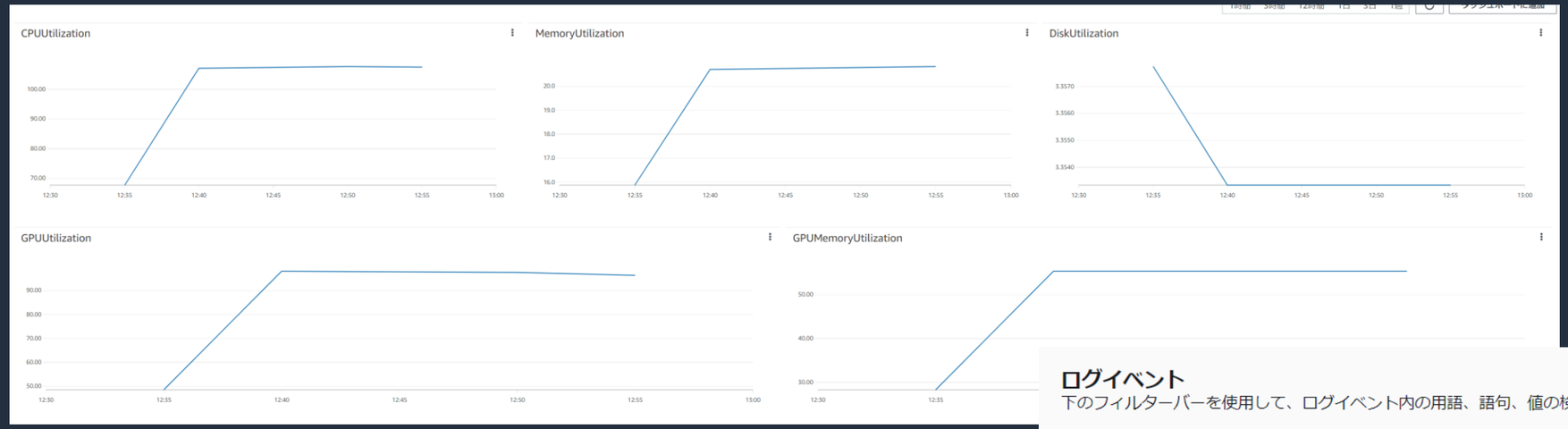
```
(略)
SM_HPS={"first-num":5,"model_dir":"s3://sagemaker-us-east-2-290000338583/tensorflow-training-2022-03-31-05-46-58-780/model","operator":"m","second-num":2}
(略)
/usr/local/bin/python3.8 hp_calc.py --first-num 5 --model_dir s3://sagemaker-us-east-2-290000338583/tensorflow-training-2022-03-31-05-46-58-780/model --operator m --second-num 2
The answer is 3
(略)
```

おしながき

- 機械学習のトレーニングで発生する課題
- Amazon SageMaker Training の概要
- 実際に動作するコードを用いたトレーニング開始方法
- Tips

ログ/メトリクス の確認

- デバッグ時には標準出力やリソースの使用状況を確認したい
- 全て自動で Amazon CloudWatch に連携され記録される



ログイベント

下のフィルターバーを使用して、ログイベント内の用語、語句、値の検索や照合ができます。 フィルターパターンの詳細

イベントをフィルター

タイムスタンプ	メッセージ
	ロードする古いイベントがあります。 さらにロードします。
2022-03-28T21:48:31.759+09:00	Steps : 330, Gen Loss Total : 47.112, Mel-Spec. Error : 0.752, s/b : 1.699
2022-03-28T21:48:39.761+09:00	Steps : 335, Gen Loss Total : 46.054, Mel-Spec. Error : 0.722, s/b : 1.698
2022-03-28T21:48:48.764+09:00	Steps : 340, Gen Loss Total : 44.757, Mel-Spec. Error : 0.749, s/b : 1.704
2022-03-28T21:48:56.766+09:00	Steps : 345, Gen Loss Total : 47.837, Mel-Spec. Error : 0.755, s/b : 1.698
2022-03-28T21:49:05.769+09:00	Steps : 350, Gen Loss Total : 46.059, Mel-Spec. Error : 0.730, s/b : 1.699
2022-03-28T21:49:13.771+09:00	Steps : 355, Gen Loss Total : 52.067, Mel-Spec. Error : 0.872, s/b : 1.693
2022-03-28T21:49:22.773+09:00	Steps : 360, Gen Loss Total : 51.251, Mel-Spec. Error : 0.774, s/b : 1.697
2022-03-28T21:49:30.778+09:00	Steps : 365, Gen Loss Total : 45.531, Mel-Spec. Error : 0.751, s/b : 1.696
2022-03-28T21:49:39.780+09:00	Steps : 370, Gen Loss Total : 48.043, Mel-Spec. Error : 0.760, s/b : 1.701
2022-03-28T21:49:47.782+09:00	Steps : 375, Gen Loss Total : 43.957, Mel-Spec. Error : 0.723, s/b : 1.702
2022-03-28T21:49:56.785+09:00	Steps : 380, Gen Loss Total : 43.076, Mel-Spec. Error : 0.741, s/b : 1.705

Git 連携

- コードの開発の作法として、あるいは MLOps の推進で Git を使うのはデファクトスタンダード
- Amazon SageMaker Training では Git リポジトリのコードをトレーニングのソースとすることをサポート

```
# Git リポジトリの設定
git_config = {'repo': 'https://github.com/username/repo-with-training-scripts.git',
              'branch': 'branch1',
              'commit': '4893e528afa4a790331e1b5286954f073b0f14a2'}

# トレーニングジョブを実行
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='mnist.py',
    py_version='py38',
    framework_version='1.9.1',
    git_config=git_config,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    role=sagemaker.get_execution_role(),
)
estimator.fit()
```

使用したいコードの
リポジトリ、ブランチ、コミットID
を dict 型で宣言

git_config 引数に宣言内容を入力

まとめ

- Amazon SageMaker Training とは？
  「用意したコード」と「用意したデータ」を「指定したコンピューティングリソース」と「用意した環境」で実行し、「実行履歴を自動記録」して「アーティファクトを自動で保存」する機能によって、モデル開発の試行錯誤を楽にするものであることを紹介

- 機械学習のコードを一切使わずに、サービスの使い方を紹介
  - トレーニングコードの持ち込み
  - データの持ち込み
  - アーティファクトの転送
  - ライブラリや独自モジュールの持ち込み
  - ハイパーパラメータを外部から入力

資料集・お問合せ・Special Thanks

AWSの日本語資料の場所: 「AWS 資料」で検索

AWSのハンズオン資料の場所: 「AWS ハンズオン」で検索

お問合せ

- 技術的なお問合せ
- 料金のお問合せ
- 個別相談会のお申込み

Special Thanks

- 音楽素材: PANICPUMPKIN様

Have a nice Amazon SageMaker Life!

aws