ML Enablement Series 【ML-Dark-04】

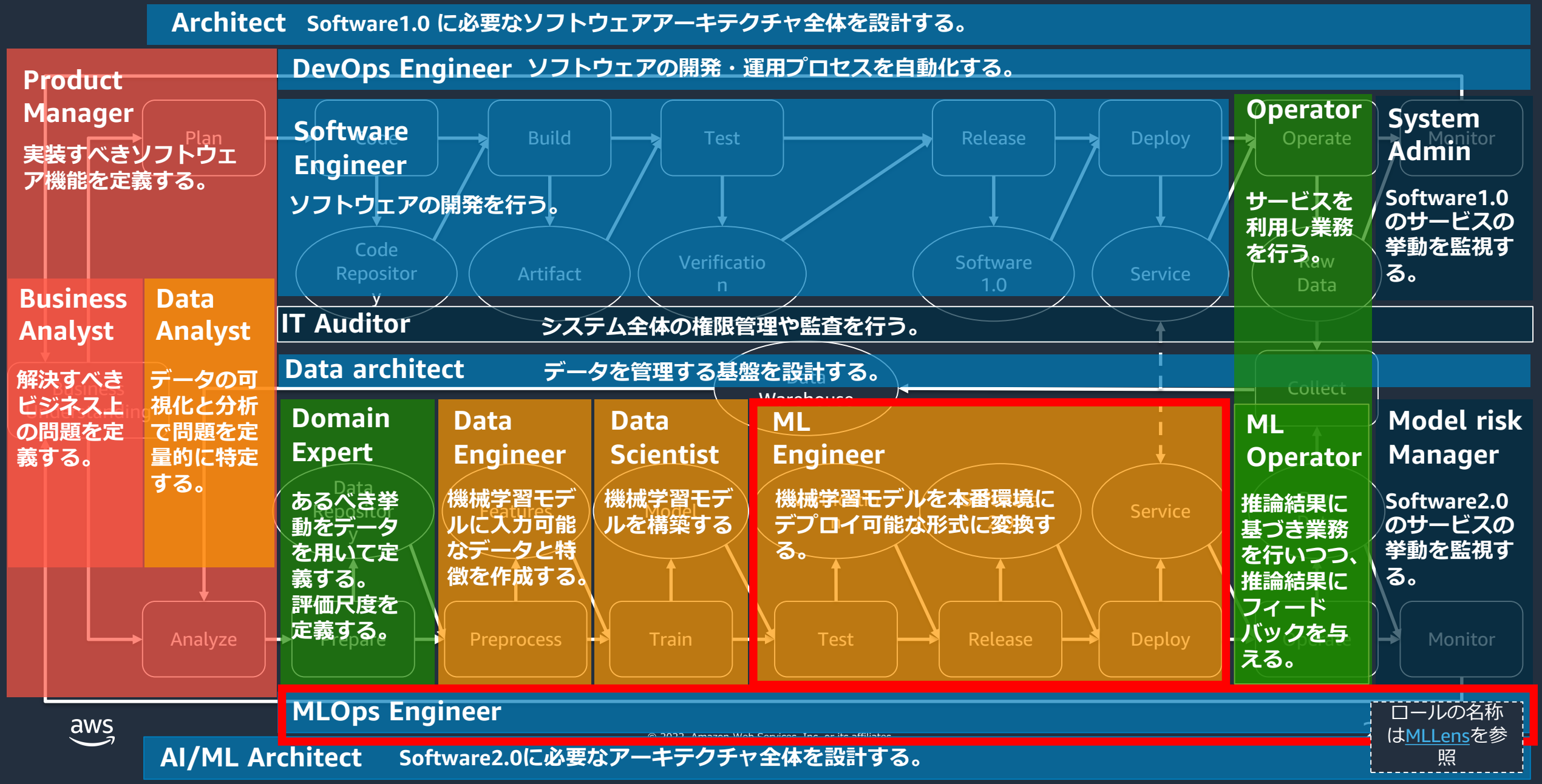
Amazon SageMaker 推論 Part2 すぐにプロダクション利用できる！ モデルをデプロイして推論する方法

AWS Black Belt Online Seminar

呉 和仁

機械学習ソリューションアーキテクト

モデルのデプロイを行う人向けの動画です



この動画のゴール

Amazon SageMaker における推論の基本機能を理解して、
セッション終了後にすぐにモデルをデプロイできるようになる

- 必要な知識
  - 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
  - デプロイ戦略
  - モデル最適化、など

前回の動画でエンジニアの皆様なら疑問に思いましたよね？

Why numpy array !?

```
response = sklearn_predictor.predict(1)
print(response,type(response))
```

```
Hello my great machine learning model for the 1st time <class 'numpy.ndarray'>
```

「リクエストが int でなぜレスポンスが numpy array になるのか」

を起点に動かし方と内部動作を解説します

登場するコード

https://github.com/aws-samples/aws-ml-jp/tree/main/sagemaker/sagemaker-inference/inference-tutorial

おしながき

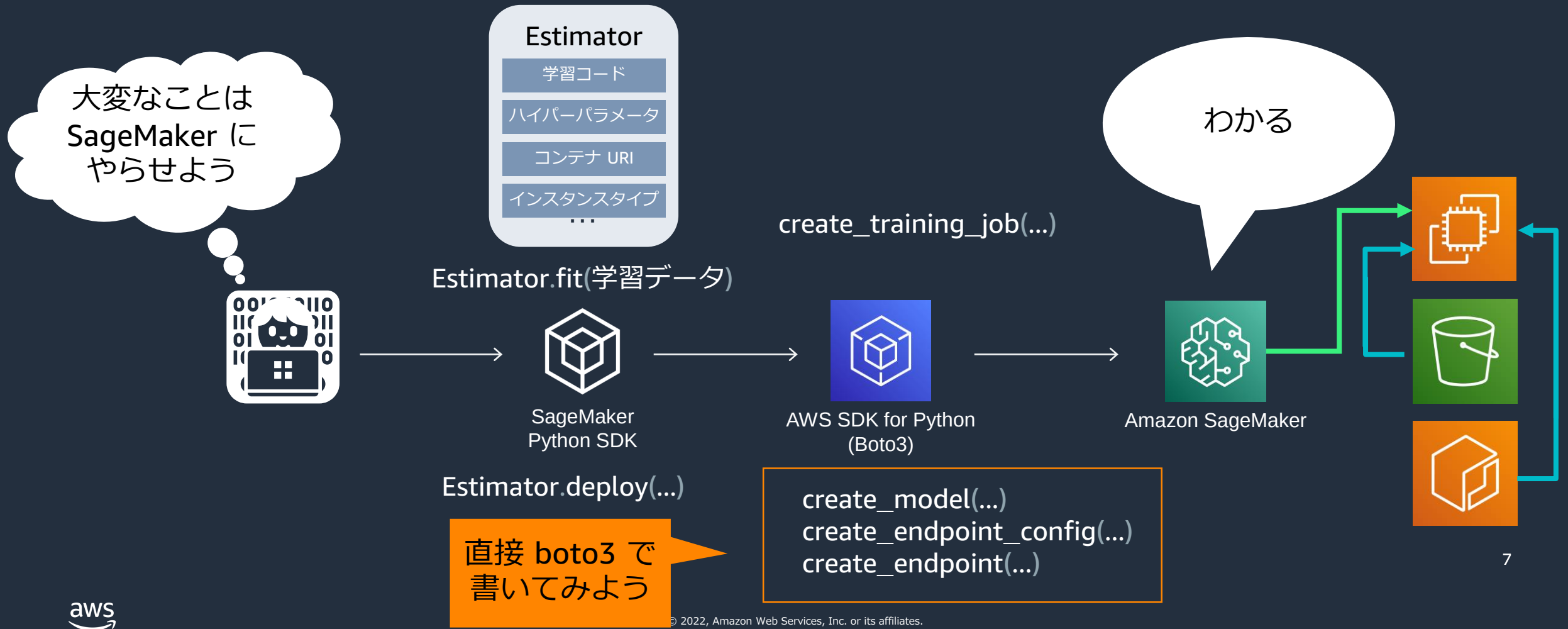
- 推論環境構築のお作法
- 前処理/後処理
- フレームワークの相違点
- 各種推論方法

おしながき

- 推論環境構築のお作法
- 前処理/後処理
- フレームワークの相違点
- 各種推論方法

Boto3 でエンドポイントを立ち上げる

SageMaker Python SDK は Boto3 のラッパーのため、
Boto3 で動かすと理解が深まる



推論エンドポイントを立てるまでの 3 ステップ

0. (モデルと推論コードの準備)

1. モデルの作成(CreateModel)
   SageMaker を用いて推論するための、モデルの推論環境を定義

2. エンドポイントコンフィグの作成(CreateEndpointConfig)
   1. で定義したモデルを動かすための、
   コンピューティングリソースや推論方式の指定

3. エンドポイントの作成(CreateEndpoint)
   リクエストを受けたら推論結果を返すようにする

0. 推論コードのお作法

```
# model.tar.gz の中に格納した my_model.txt（仮想モデル）
Hello my great machine learning model
```

```
# sourcedir.tar.gz の中に格納した infrenece.py（推論コード）
import os
def model_fn(model_dir):
    with open(os.path.join(model_dir,'my_model.txt')) as f:
        model = f.read()[:-1]
    return model
def predict_fn(input_data, model):
    response = f'{model} for the {input_data}st time'
    return response
# ------------------------------------------------------
# (本物のモデルのイメージ)
# import os, joblib
# def model_fn(model_dir):
#     model = joblib.load(os.path.join(model_dir,'model.joblib'))
#     return model
# def predict_fn(input_data, model):
#     return model.predict(input_data)
```

引数にはモデルを固めた tar.gz を展開したディレクトリ名が入る

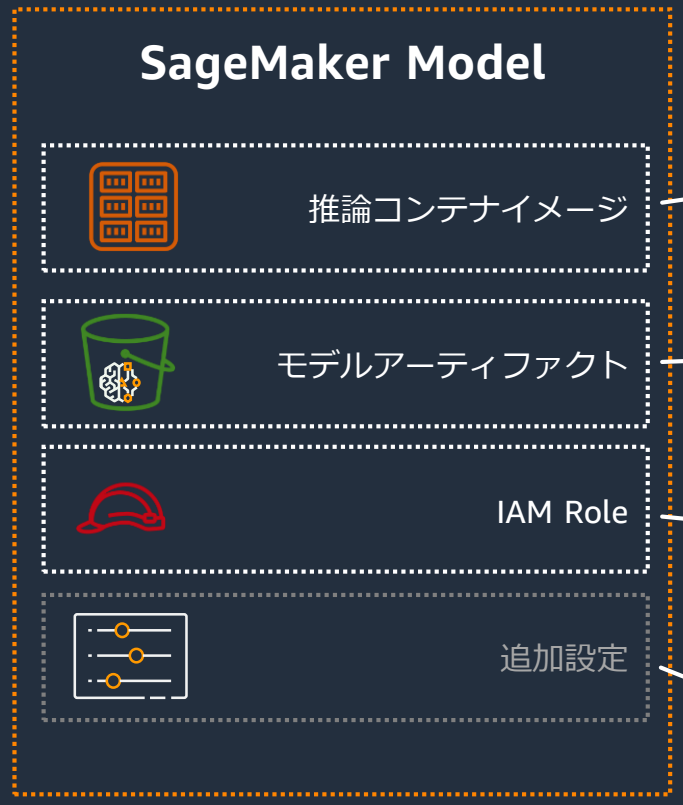
ロードしたモデルを返り値に入れると
predict_fn の第 2 引数にロードしたモデルが入る

推論結果を返り値に入れる

joblib や pickle で dump した（あるいは .pt ファイルなど)
モデルを読むコードを書く

リクエストデータ

1. モデルの作成: 概念



推論に必要な情報を登録

推論用コンテナイメージが格納されている
ECR などのコンテナイメージの URI

学習済みモデルが格納された S3 パス

デプロイ時にモデルアーティファクトや
docker イメージにアクセスするための IAM role

追加設定はデプロイの種類によって変わる。
例）VPC 設定、マルチコンテナデプロイ、
マルチモデルデプロイ

1. モデル作成: コード

```
response = sm_client.create_model(
  ①ModelName=model_name, ExecutionRoleArn=role,②
    PrimaryContainer={
      ③'Image': container_image_uri, 'ModelDataUrl': model_s3_uri,④
        'Environment': {
          ⑤'SAGEMAKER_CONTAINER_LOG_LEVEL': '20','SAGEMAKER_PROGRAM': 'inference.py',⑥
            'SAGEMAKER_REGION': region, 'SAGEMAKER_SUBMIT_DIRECTORY': source_s3_uri}
    },                                ⑦                                    ⑧
)
```

モデル設定

名前
SKLearnModel
①アカウント・リージョン内で一意の名前

ARN
arn:aws:sagemaker:{region}:{account}:model/sklearnmodel

作成時刻
Jul 21, 2022 12:49 UTC

IAM ロール ARN
②推論環境にアタッチするロール
arn:aws:iam::{account}:role/service-role/AmazonSageMaker-ExecutionRole-{YYYY}{MM}{DD}T{hh}{mm}{ss}

コンテナ 1

コンテナ名
Container 1

イメージ
③推論に使用するコンテナイメージ
683313688378.dkr.ecr.{region}.amazonaws.com/sagemaker-scikit-learn:1.0-1-cpu-py3

環境変数

モデルデータの場所
④推論モデル
s3://sagemaker-{region}-{account}/hello_sagemaker_inference/model.tar.gz

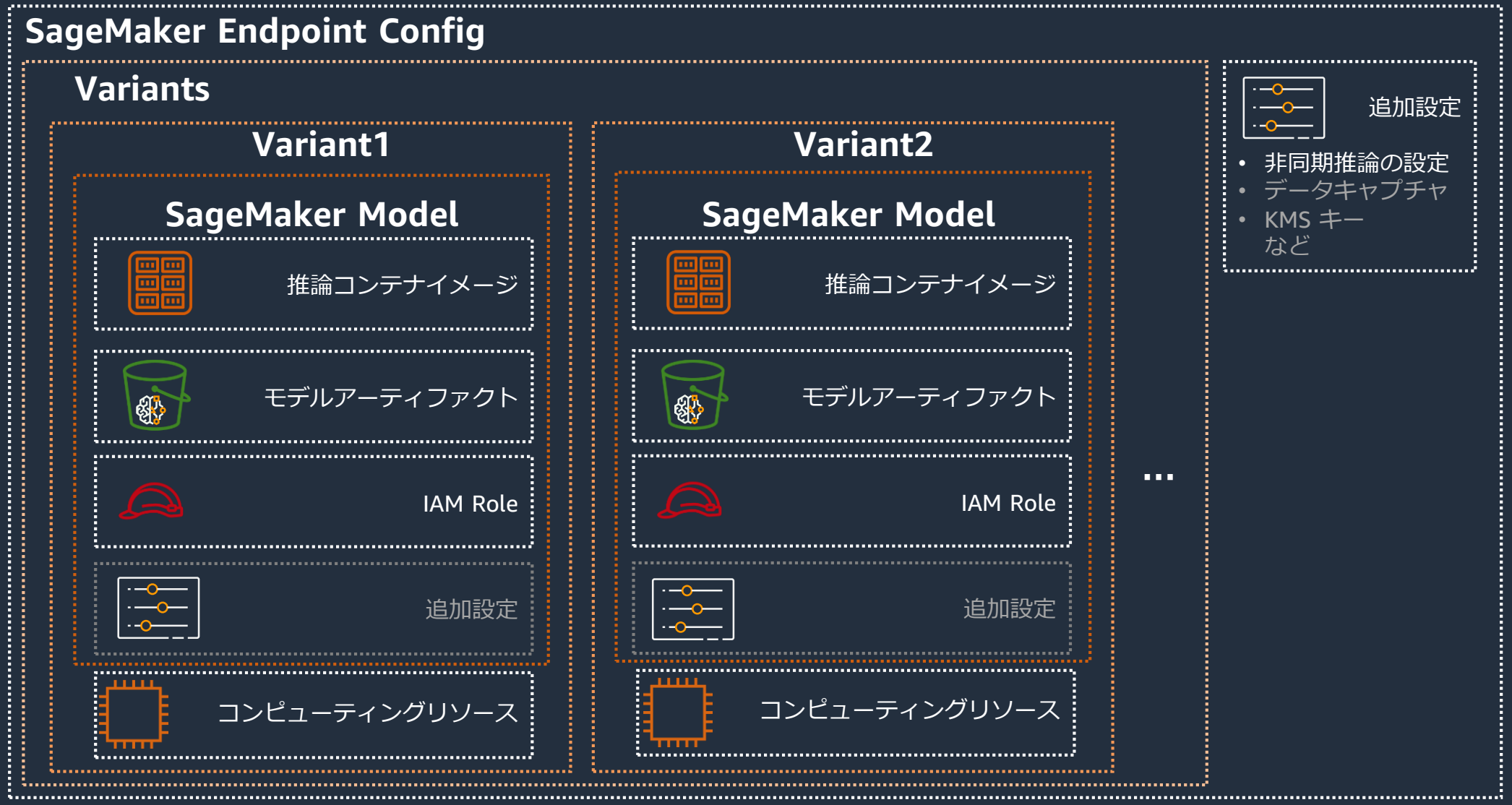
Mode
単一モデル

SageMaker SDK から推論エンドポイントを立ち上げた時のモデルの設定

キー	値
SAGEMAKER_CONTAINER_LOG_LEVEL	20 ⑤ログレベル
SAGEMAKER_PROGRAM	inference.py ⑥推論コードのファイル名
SAGEMAKER_REGION	{region} ⑦リージョン
SAGEMAKER_SUBMIT_DIRECTORY	s3://sagemaker-{region}-{account}/hello_sagemaker_inference/sourcedir.tar.gz ⑧推論コードの S3 URI

2. エンドポイントコンフィグの作成: 概念

エンドポイントに必要な情報を登録する



エンドポイントコンフィグ作成のコード

```
response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name, ①
    ProductionVariants=[    ②                          ③
        {'ModelName': model_name, 'VariantName': 'AllTrafic',
         'InstanceType': 'ml.m5.large', 'InitialInstanceCount': 1},
    ]                              ④                          ⑤
)
```

エンドポイント設定の指定

名前	ARN	暗号化キー	作成時刻
SKLearnModel ①アカウント・リージョン内で一意の名前	arn:aws:sagemaker:{region}:{account}:endpoint-config/sklearnmodel	-	Jul 21, 2022 13:24 UTC

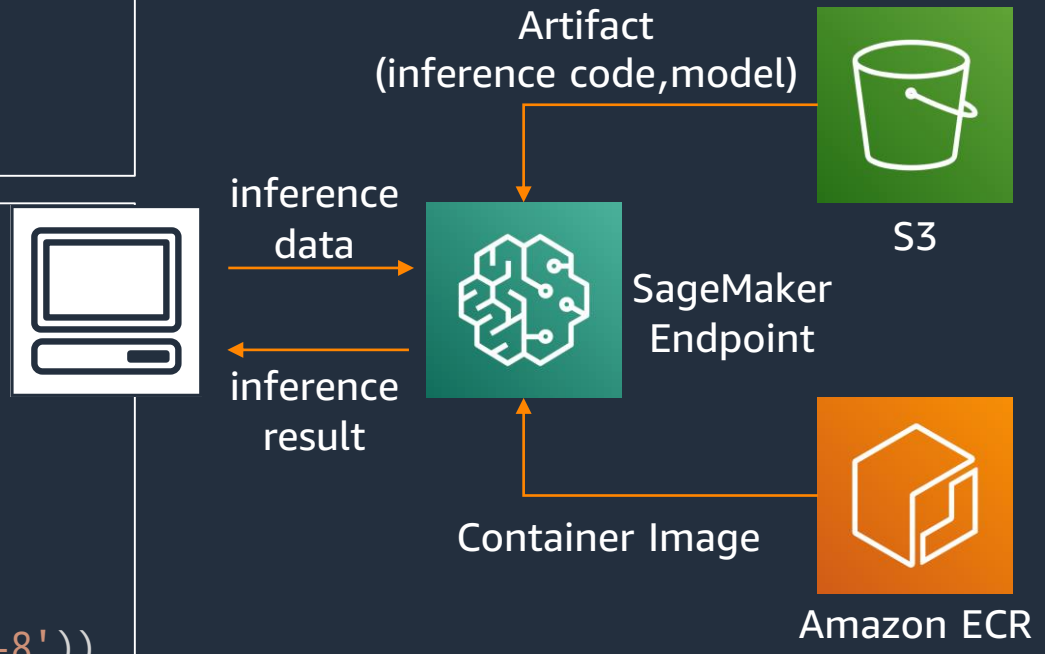
本番稼働用バリアント

モデル名	トレーニングジョブ	バリアント名	インスタンスタイプ	Elastic Inference	初期インスタンス数	初期重量
②作成済のモデルの名前 SKLearnModel	-	③ AllTraffic	④ ml.m5.large	-	⑤ 1	1

モデルのデプロイと推論

```
# デプロイ
response = sm_client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name,
)
```

```
# リクエスト
response = smr_client.invoke_endpoint(
    EndpointName=endpoint_name,
    ContentType='application/json',
    Accept='application/json',
    Body='1'
)
# レスポンス確認
predictions = json.loads(response['Body'].read().decode('utf-8'))
print(predictions)
```



今度は小数点！？

```
Hello my great machine learning model for the 1.0st time
```

おしながき

- 推論環境構築のお作法
- 前処理/後処理
- フレームワークの相違点
- 各種推論方法

前処理/後処理のデフォルトで通るコードがある

エンドポイント起動時に model_fn でモデルを読み込み、
以降、推論イベント発生時には

1. input_fn(前処理)
2. predict_fn(予測)
3. output_fn(後処理)

の順番で実行される。

xxx_fn が設定されていれば xxx_fn を使い、
設定されていなければ default_xxx_fn を使う

```
# https://github.com/aws/sagemaker-scikit-learn-container/blob/master/
src/sagemaker_sklearn_container/serving.py より抜粋
def _user_module_transformer(user_module):
    model_fn = getattr(user_module, 'model_fn', default_model_fn)
    input_fn = getattr(user_module, 'input_fn', default_input_fn)
    predict_fn = getattr(user_module, 'predict_fn', default_predict_fn)
    output_fn = getattr(user_module, 'output_fn', default_output_fn)

    return transformer.Transformer(model_fn=model_fn, input_fn=input_fn, predict_fn=predict_fn,output_fn=output_fn)
```

デフォルトの前処理 (default_input_fn)

```
# https://github.com/aws/sagemaker-scikit-learn-container/blob/master/
src/sagemaker_sklearn_container/handler_serving.py より抜粋
def default_input_fn(input_data, content_type):
    np_array = encoders.decode(input_data, content_type)
    return np_array.astype(np.float32) if content_type in content_types.UTF8_TYPES else np_array
```

```
# https://github.com/aws/sagemaker-inference-toolkit/blob/master/
src/sagemaker_inference/decoder.pyより抜粋
def decode(obj, content_type):
    try:
        decoder = _decoder_map[content_type]
        return decoder(obj)
    except KeyError:
        raise errors.UnsupportedFormatError(content_type)
_decoder_map = {
    content_types.NPY: _npy_to_numpy,
    content_types.CSV: _csv_to_numpy,
    content_types.JSON: _json_to_numpy,
    content_types.NPZ: _npz_to_sparse,
}
def _json_to_numpy(string_like, dtype=None):  # type: (str) -> np.array
    data = json.loads(string_like)
    return np.array(data, dtype=dtype)
```

application/json の場合
リクエストデータを 32bit floatの
numpy arrayとして解釈する

デフォルトの後処理(default_output_fn)

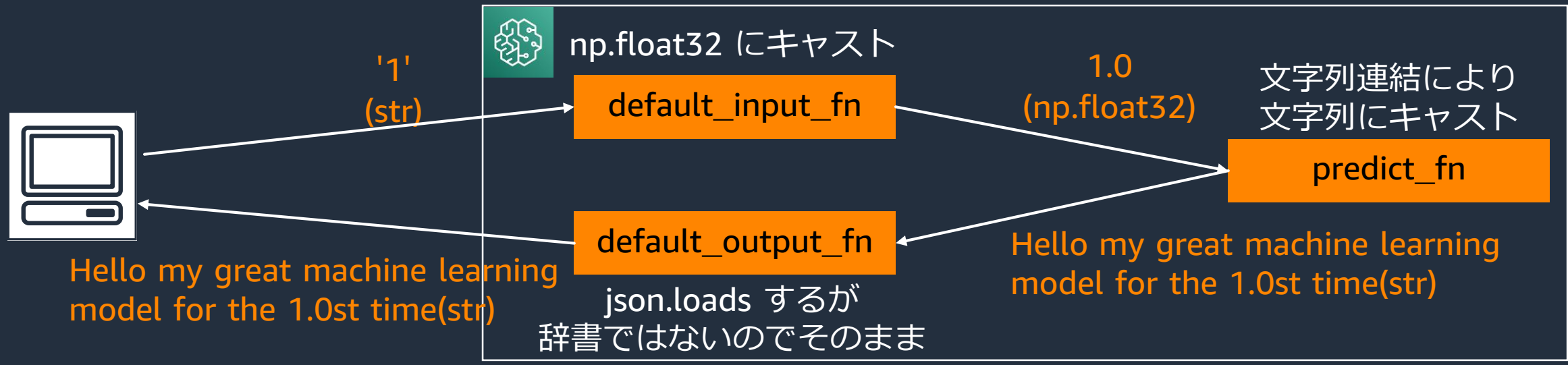
```
# https://github.com/aws/sagemaker-scikit-learn-container/blob/master/
src/sagemaker_sklearn_container/handler_serving.py より抜粋
def default_output_fn(prediction, accept):
    return encoders.encode(prediction, accept), accept
```

```
# https://github.com/aws/sagemaker-inference-toolkit/blob/master/
src/sagemaker_inference/encoder.pyより抜粋
def encode(array_like, content_type):
    try:
        encoder = _encoder_map[content_type]
        return encoder(array_like)
    except KeyError:
        raise errors.UnsupportedFormatError(content_type)
_encoder_map = {
    content_types.NPY: _array_to_npy,
    content_types.CSV: _array_to_csv,
    content_types.JSON: _array_to_json,
}
def _array_to_json(array_like):
    def default(_array_like):
        if hasattr(_array_like, "tolist"):
            return _array_like.tolist()
        return json.JSONEncoder().default(_array_like)
    return json.dumps(array_like, default=default)
```

Accept='application/json'の場合
json 文字列にして返す

つまり

default_input_fn が浮動小数(np.float32)で解釈したあと
文字列に連結したため、小数が出現した



なぜ SageMaker SDK で predict すると整数だったのか

predictor に Numpy の Serializer がデフォルトで入り、整数の numpy array としてリクエストされ、context_type='application/x-npy'でリクエストし、そのまま numpy array として処理されるため

```
response = sklearn_predictor.predict(1)
print(response, type(response))
```

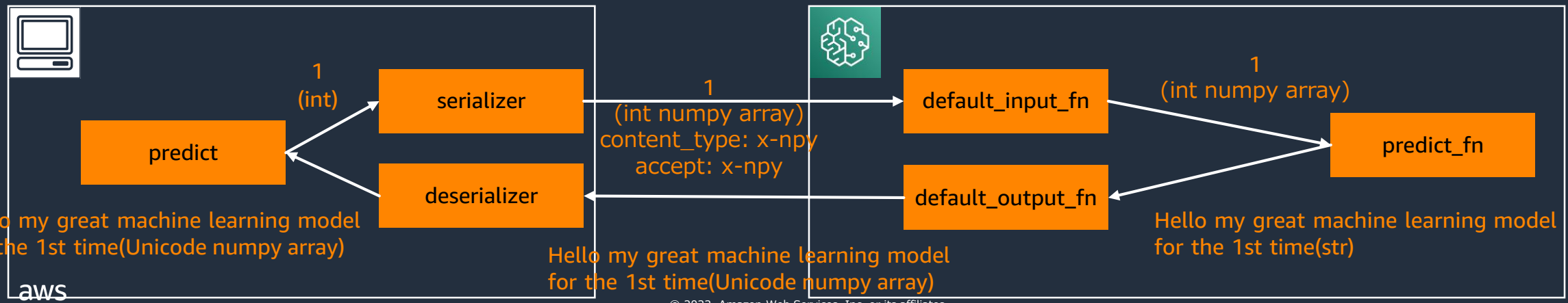
```
Hello my great machine learning model for the 1st time <class 'numpy.ndarray'>
```

```
sklearn_predictor.serializer, sklearn_predictor.deserializer
```

```
(<sagemaker.serializers.NumpySerializer at 0x7f91356ca8b0>,
 <sagemaker.deserializers.NumpyDeserializer at 0x7f91356ca910>)
```

```
# https://github.com/aws/sagemaker-python-sdk/blob/master/
src/sagemaker/serializers.py より抜粋
class NumpySerializer(SimpleBaseSerializer):
    def serialize(self, data):
        return self._serialize_array(np.array(data))
    def _serialize_array(self, array):
        buffer = io.BytesIO()
        np.save(buffer, array)
        return buffer.getvalue()
```

```
# https://github.com/aws/sagemaker-inference-toolkit/blob/master/
src/sagemaker_inference/encoder.py より抜粋 (default_input_json から呼ばれる)
def _array_to_npy(array_like):
    buffer = BytesIO()
    np.save(buffer, array_like)
    return buffer.getvalue()
```



ちなみに default_model_fn は必ずエラーが発生

ユーザーが model_fn を用意する必要がある

```
# https://github.com/aws/sagemaker-scikit-learn-container/blob/master/
src/sagemaker_sklearn_container/handler_service.py より抜粋

def default_model_fn(model_dir):
    """Loads a model. For Scikit-learn, a default function to load a model is not provided.
    Users should provide customized model_fn() in script.
    Args:
        model_dir: a directory where model is saved.
    Returns: A Scikit-learn model.
    """
    raise NotImplementedError(textwrap.dedent("""
    Please provide a model_fn implementation.
    See documentation for model_fn at https://github.com/aws/sagemaker-python-sdk
    """))
```

model_fn を用意しなさい というコメント

raise するだけ

default_predict_fn はそのまま使えるなら使ってもよい

一般的な Scikit-Learn のモデルならば変更不要

```
# https://github.com/aws/sagemaker-scikit-learn-container/blob/master/
src/sagemaker_sklearn_container/handler_service.py より抜粋

def default_predict_fn(input_data, model):
    """A default predict_fn for Scikit-learn. Calls a model on data deserialized in input_fn.
    Args:
        input_data: input data (Numpy array) for prediction deserialized by input_fn
        model: Scikit-learn model loaded in memory by model_fn
    Returns: a prediction
    """
    output = model.predict(input_data)
    return output
```

Scikit-Learn で作った
モデルで推論するコード

自前の前処理/後処理実装例 (inference.py)

```python
# モデル読み込み
def model_fn(model_dir):
    with open(os.path.join(model_dir,'my_model.txt')) as f:
        hello = f.read()[:-1]
    return hello
# 前処理
def input_fn(input_data, content_type):
    if content_type == 'text/csv':
        transformed_data = input_data.split(',')
    else:
        raise ValueError("Illegal content type")
    return transformed_data
```

- どんな数値が来ても正しい序数を返す
- text/csv のみ受け付ける
- リクエストは複数の数値を受け付ける
- レスポンスは改行で区切る

```python
# 予測
def predict_fn(transformed_data, model):
    prediction_list = []
    for data in transformed_data:
        if data[-1] == '1':
            ordinal = f'{data}st'
        elif data[-1] == '2':
            ordinal = f'{data}nd'
        elif data[-1] == '3':
            ordinal = f'{data}rd'
        else:
            ordinal = f'{data}th'
        prediction = f'{model} for the {ordinal} time'
        prediction_list.append(prediction)
    return prediction_list
# 後処理
def output_fn(prediction_list, accept):
    if accept == 'text/csv':
        response = ''
        for prediction in prediction_list:
            response += prediction + '¥n'
    else:
        raise ValueError("Illegal accept type")
    return response, accept
```

自前の前処理/後処理の推論結果

リクエストとレスポンスの制御ができた！

```
response = smr_client.invoke_endpoint(
    EndpointName=endpoint_name,
    ContentType='text/csv',
    Accept='text/csv',
    Body='1,2,3,10000'
)
predictions = response['Body'].read().decode('utf-8')
print(predictions)
```

```
Hello my great machine learning model for the 1st time
Hello my great machine learning model for the 2nd time
Hello my great machine learning model for the 3rd time
Hello my great machine learning model for the 10000th time
```

おしながき

- 推論環境構築のお作法
- 前処理/後処理
- フレームワークの相違点
- 各種推論方法

Scikit-Learn コンテナと PyTorch コンテナの主な違い

推論コードの格納先と、create_model に違い

- Scikit-Learn コンテナは推論コードだけを tar.gz (e.g. sourcedir.tar.gz)に格納したが、PyTorch コンテナはモデルと一緒(e.g. model.tar.gz)に格納
- SAGEMAKER_SUBMIT_DIRECTORY に割り当てる値が、Scikit-Learn コンテナは sourcedir.tar.gz の S3 URI だが PyTorch コンテナは 推論コードが展開されるディレクトリ
  ※PyTorch v1.2 以降のコンテナが対象

```
# Scikit-Learn
response = sm_client.create_model(
    ModelName=model_name,
    PrimaryContainer={
        'Image': container_image_uri,
        'ModelDataUrl': model_s3_uri,
        'Environment': {
            'SAGEMAKER_CONTAINER_LOG_LEVEL': '20',
            'SAGEMAKER_PROGRAM': 'inference.py',
            'SAGEMAKER_REGION': region,
            'SAGEMAKER_SUBMIT_DIRECTORY': source_s3_uri}
    },
    ExecutionRoleArn=role,
)
```

```
# PyTorch
response = sm_client.create_model(
    ModelName=model_name,
    PrimaryContainer={
        'Image': container_image_uri,
        'ModelDataUrl': model_s3_uri,
        'Environment': {
            'SAGEMAKER_CONTAINER_LOG_LEVEL': '20',
            'SAGEMAKER_PROGRAM': 'inference.py',
            'SAGEMAKER_REGION': region,
            'SAGEMAKER_SUBMIT_DIRECTORY': '/opt/ml/model/code'}
    },
    ExecutionRoleArn=role,
)
```

具体的な使い方例: https://github.com/aws-samples/aws-ml-jp/tree/main/sagemaker/sagemaker-inference/pytorch

TensorFlow コンテナ

Scikit-Learn コンテナや PyTorch コンテナとは作りが少し異なる

- TensorFlow のコンテナでは TensorFlow Serving をベースに作ってあるため、saved_model 形式で保存したモデルを model.tar.gz に固めて保存する
- 前処理/後処理は input_handler/output_handler で記述

具体的な使い方例
https://github.com/aws-samples/aws-ml-jp/tree/main/sagemaker/sagemaker-inference/tensorflow

SageMaker Docs
https://sagemaker.readthedocs.io/en/stable/frameworks/tensorflow/using_tf.html#deploy-tensorflow-serving-models

TensorFlow Serving Docs
https://www.tensorflow.org/tfx/tutorials/serving/rest_simple

おしながき

- 推論環境構築のお作法
- 前処理/後処理
- フレームワークの相違点
- 各種推論方法

リアルタイム推論
SageMaker Python SDK と boto3 の違い

```python
# SageMaker Python SDK
sklearn_model = SKLearnModel(
    name = model_name,
    model_data=model_s3_uri,
    role= role,
    framework_version = '1.0-1',
    py_version='py3',
    entry_point='inference.py',
    source_dir=f'./{source_dir}/'
)
sklearn_predictor = sklearn_model.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.large',
    enpoint_name=endpoint_name
)
```

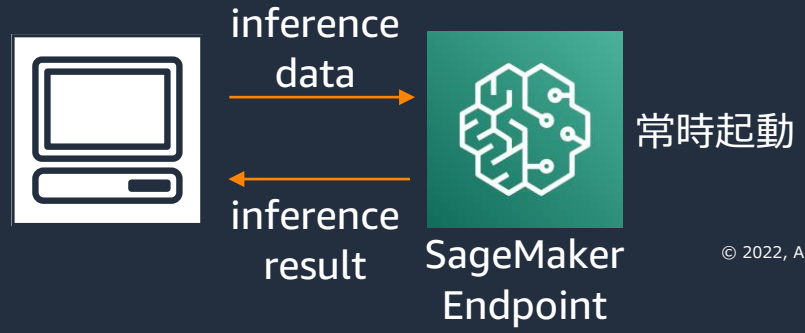
requirementst.txt
を入れられる

≒

```python
# Boto3
response = sm_client.create_model(
    ModelName=model_name,
    PrimaryContainer={
        'Image': container_image_uri,
        'ModelDataUrl': model_s3_uri,
        'Environment': {
            'SAGEMAKER_CONTAINER_LOG_LEVEL': '20',
            'SAGEMAKER_PROGRAM': 'inference.py',
            'SAGEMAKER_REGION': region,
            'SAGEMAKER_SUBMIT_DIRECTORY': source_s3_uri}
    },ExecutionRoleArn=role,
)
response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[{
            'VariantName': 'AllTrafic',
            'ModelName': model_name,
            'InitialInstanceCount': 1,
            'InstanceType': 'ml.m5.large',
    }],
)
response = sm_client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name,
)
```

requirementst.txt
を入れられる

※boto3 では別途下記操作が必要
- 推論コードをtar.gzに固めてS3にアップロード
- コンテナイメージのURIを取得



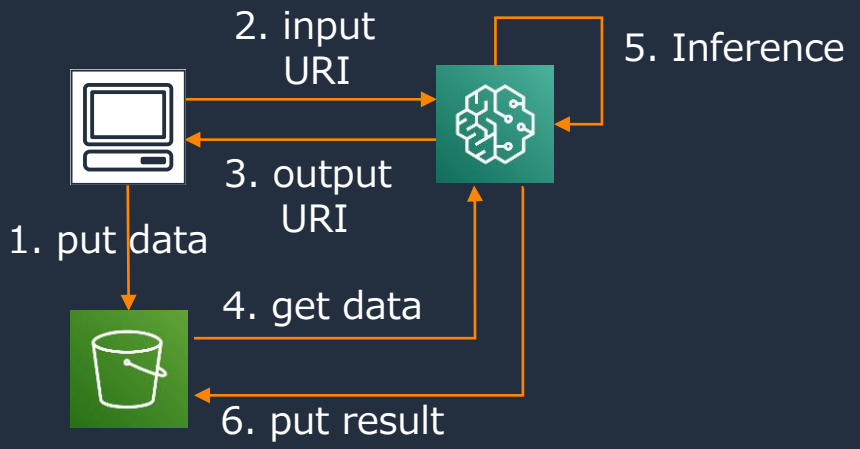
非同期推論

リアルタイム推論との違いは create_endpoint_config と 推論 API

※create_model, create_endpoint はリアルタイム推論と同じのため省略

```
# EndpointConfig 作成
response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[
        {
            'VariantName': 'AllTrafic',
            'ModelName': model_name,
            'InitialInstanceCount': 1,
            'InstanceType': 'ml.m5.large',
        },
    ],
    AsyncInferenceConfig={
        "OutputConfig": {
            "S3OutputPath": f"s3://{bucket}/{prefix}"
        },
    }
)
```

推論結果の格納先を指定



invoke_endpoint_async API を使い、
InputLocation 引数で推論データを指定する

```
# 推論
response = smr_client.invoke_endpoint_async(
    EndpointName=endpoint_name,
    InputLocation=input_data_s3_uri,
    ContentType='text/csv',
    Accept='text/csv',
)
```

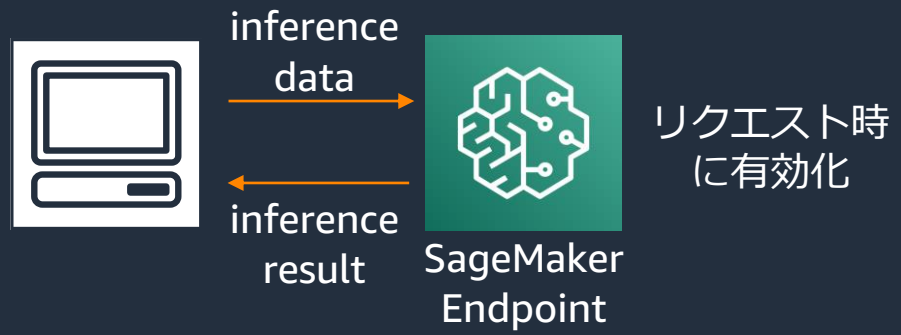
```
# 推論結果取得
output_s3_uri = response['OutputLocation']
output_key = output_s3_uri.replace(f's3://{bucket}/','')
obj = s3_client.get_object(Bucket=bucket, Key=output_key)
predictions = obj['Body'].read().decode()
print(predictions)
```

サーバーレス推論

リアルタイム推論との違いは endpoint_config の設定のみ

※create_model, create_endpoint, invoke_endpoint はリアルタイム推論と同じのため省略

```
# EndpointConfig 作成
response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[
        {
            'ModelName': model_name,
            'VariantName': 'AllTrafic',
            'ServerlessConfig': {
                'MemorySizeInMB': 1024,
                'MaxConcurrency': 3
            }
        },
    ],
)
```



variant に、サーバーレスエンドポイントに
割り当てるメモリと、最大同時呼び出し数を設定

バッチ推論

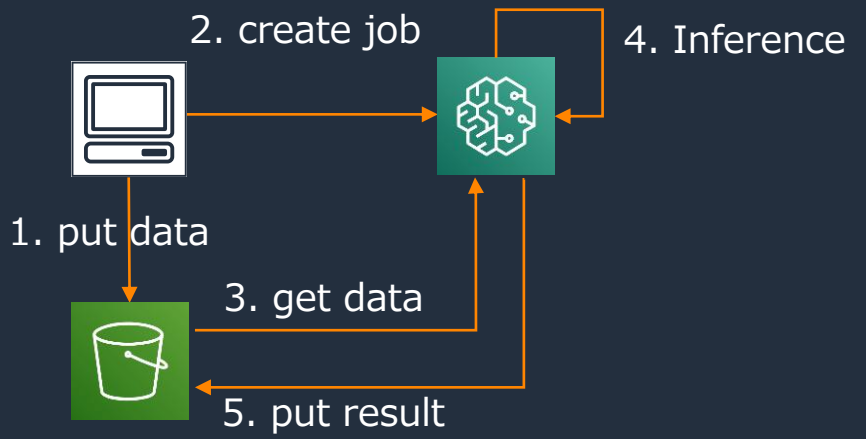
endpoint_config 以降は不要で、create_transform_job で起動

※create_model はリアルタイム推論と同じのため省略

```
transform_job_name: Final[str] = f'{model_name}TransformJob-{uuid4()}'
print(transform_job_name)
response = sm_client.create_transform_job(
    TransformJobName=transform_job_name,
    ModelName=model_name,
    TransformInput={
        'DataSource': {
            'S3DataSource': {
                'S3DataType': 'S3Prefix',
                'S3Uri': f's3://{bucket}/{input_prefix}'
            }
        },
        'ContentType': 'text/csv',
    },
    TransformOutput={
        'S3OutputPath': f's3://{bucket}/{output_prefix}',
        'Accept': 'text/csv',
    },
    TransformResources={
        'InstanceType': 'ml.m5.large',
        'InstanceCount': 1,
    }
)
```

推論データの格納先を指定

推論結果の出力先を指定




```
# 推論結果取得
content  = s3_client.list_objects_v2(Bucket=bucket,Prefix=output_prefix)['Contents']
for content in content :
    obj = s3_client.get_object(Bucket=bucket, Key=content['Key'])
    predictions = obj['Body'].read().decode()
    print(predictions)
```

まとめ

- Scikit-Learn コンテナのリアルタイム推論を題材に Boto3 をベースに model, endpoint config, endpoint を作成することで、推論を開始できることを解説

- 前処理/後処理を追加してリクエストとレスポンスの形式をコントロールできることを解説

- コンテナごとに使い方が違うことを紹介

- リアルタイム推論、非同期推論、サーバーレス推論、バッチ推論の実行の仕方とその違いについて解説

ML Enablement Seriesの動画

機械学習モデルをビジネス価値につなげる方法を全力で解説！

Light Part

製品やサービスに機械学習を導入するプロジェクトの進め方

https://bit.ly/3M1F9as

Step Up!!

Dark Part

機械学習モデルの開発や運用をマネージドサービスで効率的に行う方法

https://bit.ly/3927PCN

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

Thank you!