mixi
GROUP

カスタマーサポートにおける Amazon SageMakerを活用した 自然言語処理・MLシステム構築

第10回 Amazon SageMaker 事例祭り
2019年11月28日

株式会社ミクシィ
本間 光宣

自己紹介

- 本間 光宣
- 株式会社ミクシィ
  - 統括管理本部CS部CREグループCREチーム
  - 2018年新卒
- 機械学習は社会人になってから学習を始める

CREの紹介

- Customer Reliability Engineer
- 顧客信頼の最大化を目的としたエンジニア
- ミクシィのCREは、技術でCSをサポートし顧客信頼の最大化を目指している
- 具体的な業務
  - CS向け管理ツールの運用保守
  - 問い合わせ起因の技術的調査
  - 健全化のためのツール開発
  - CS業務の効率化・自動化

目次

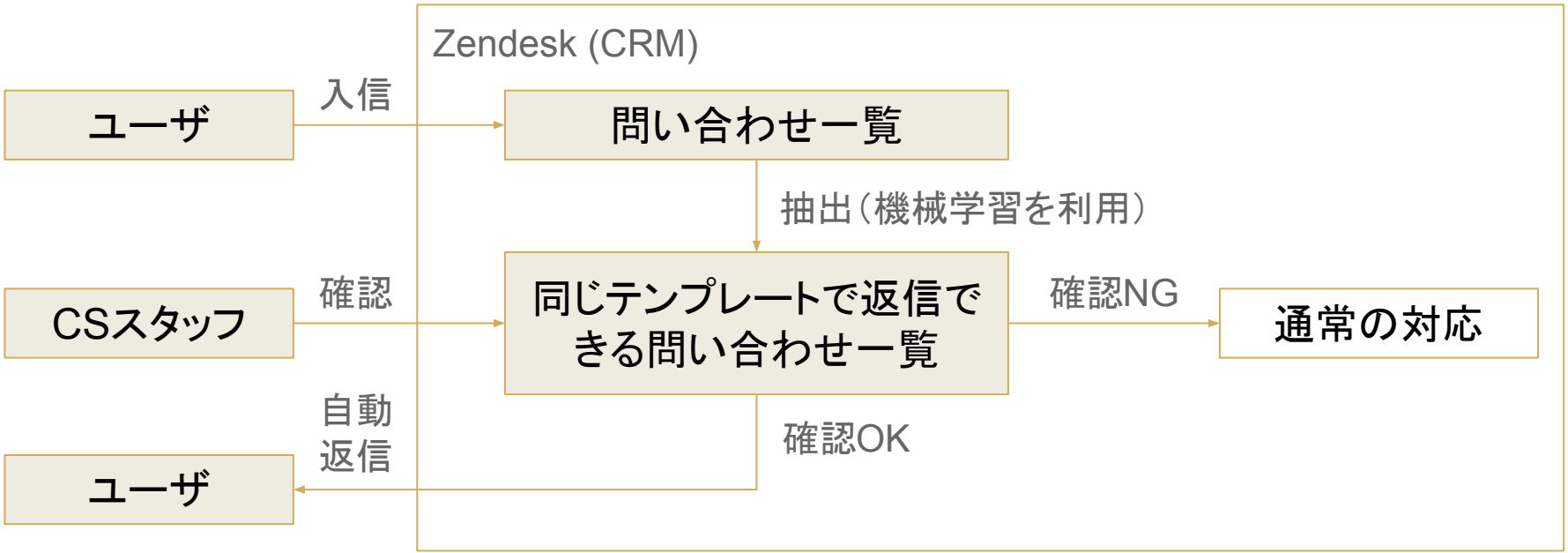
1. 一括返信システムについて
2. SageMakerを用いた学習について
3. SageMakerと他のサービスとの連携について

mixi
GROUP

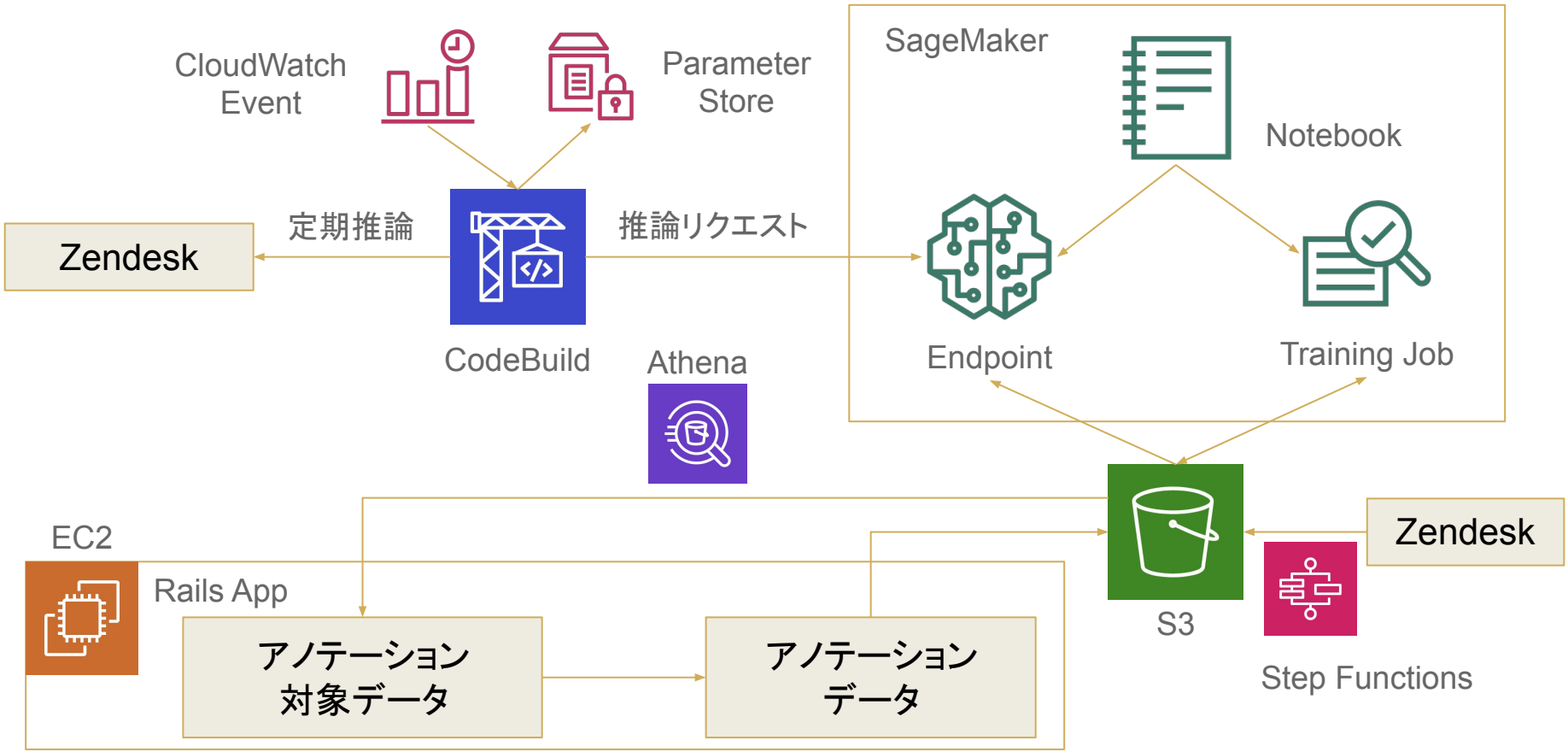
一括返信システムについて

一括返信システムの概要

- 同じテンプレートで返信できる問い合わせを一括で返信するシステム
- 間違い返信はしたくないので完全自動化はせず確認を挟むフローに
- チームで初めての機械学習を用いたシステム

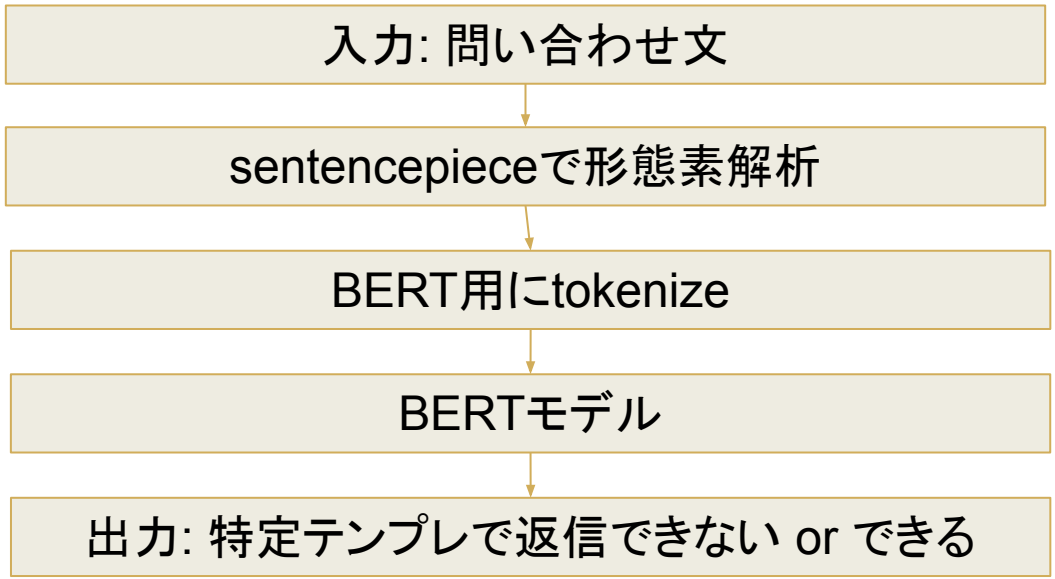


一括返信システムの構成



機械学習モデルについて

- BERTでFine tuningをおこなう
  - 日本語 Wikipedia で学習された公開されているモデルを拝借（yoheikikuta/bert-japanese）



課金したのにオーブが増えない！

課金, した, の, に, オー, ブ, が増え, ない

[CLS] 4 1 7 2 9 10 3 5 [SEP]

0 or 1

データの前処理

- モデルの精度を上げるためにテキストの前処理をおこなう
- 全角半角等を統一、絵文字の削除など
- 複雑な正規化は一切していない（独自辞書の利用など）

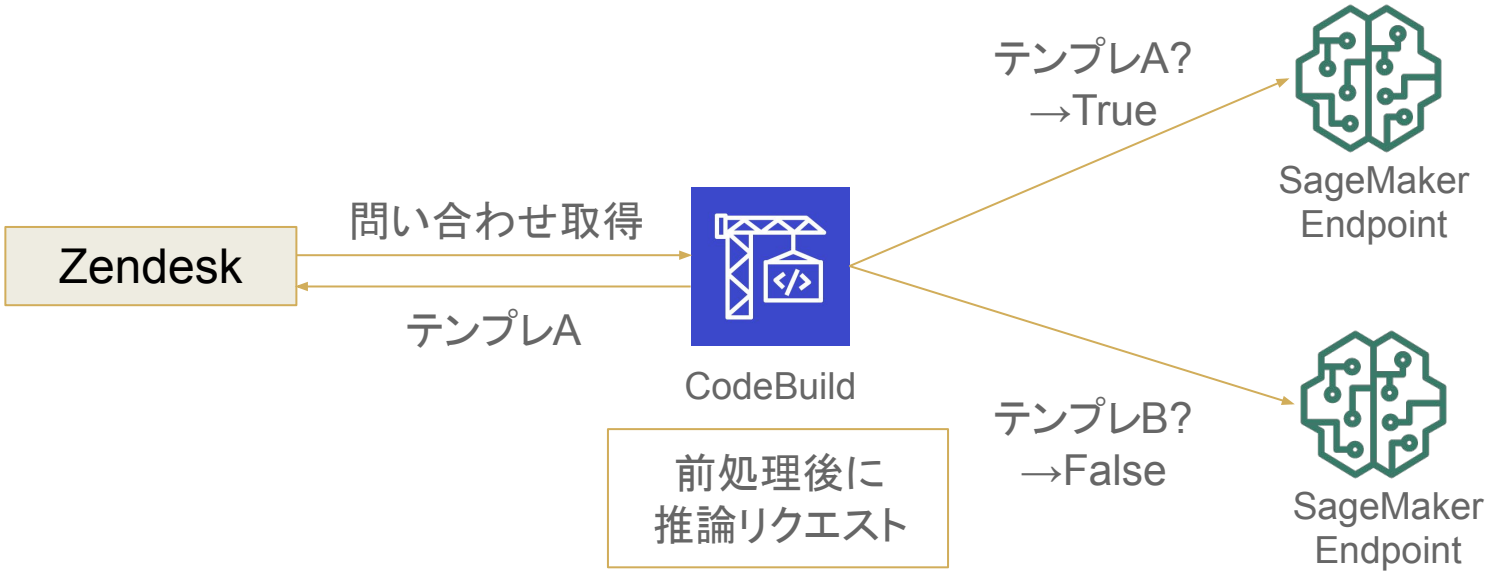
【ID】
xxxxx
【端末】
iPhone X
【本文】
課金したのにｵｰﾌﾞが増え
ない！！！💢💢💢

課金したのにｵｰﾌﾞが増え
ない！！！💢💢💢

課金したのにオーブが増え
ない！

複数の二値分類器を用いた推論

- 複数の二値分類器を用いて多値分類をおこなっている
- 問い合わせの前処理はCodeBuild側でおこない、
  Endpointは前処理済みのデータに対して推論結果を返すだけ



mixi
GROUP

SageMakerを用いた学習について

学習の流れ

- 過去の問い合わせをアノテーションし学習データとして利用

学習環境について

- 最初からSageMakerを利用したのではなくまずはGPUマシンで学習
- 本格的に進めると決めた段階でGPUマシンからAWSに移行した

- コンセプト検証
- コスト優先でまずは
  ローカルで検証

- 本格的に学習
- チームメンバーが
  慣れているAWSに移行
- SageMakerを含む
  様々なサービスを活用

学習に用いた技術

最初の構成

現在の構成

Zendesk

過去の問い合わせデータ

前処理済みデータ

アノテーション対象データ

アノテーションデータ

学習済みモデル

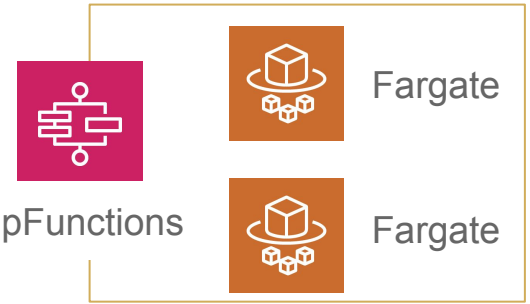
GPUマシン

GPUマシン

GPUマシン

SpreadSheetでCSスタッフにアノテーションを依頼

GPUマシン



Athena

SageMaker

GPUマシンを用いた学習について

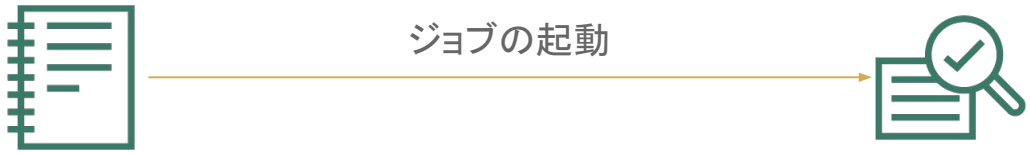
- 環境構築
  - GPUを使うために各種ドライバをインストール
  - Python環境はpipenvで作成
- 学習
  - リモートデスクトップ + Jupyter Notebookで学習を実行
- 問題点・詰まった点
  - 共用PCなのでチームで1つのタスクしか実行できない
  - 初めてのGPUの環境構築
  - 個人的にはWindowsに慣れていないのでWindowsに躓く・・

SageMakerを用いた学習について

- SageMakerで学習をするためには以下の3(4)点セットが必要
- SageMakerを用いた学習の流れ
  - 学習コードをNotebookインスタンスに置く
  - 学習に使うデータをS3に置く
  - （Training Jobで使うイメージをECRにpushしておく）
  - Training Jobを作成するJupyter Notebookを実行する

Training Jobを用いた学習の流れ

1. Jupyter Notebookからイメージを指定してジョブを作成
2. ジョブ作成時に指定されるentry_point, source_dirをTraining Jobに転送
3. TrainingJobではsource_dir以下にrequirements.txtがあればインストール
4. entry_pointのmain関数を実行して学習を開始
5. 学習後は指定場所に保存されたデータをS3に保存してジョブは終了



Notebook Instance

Training Job

以下のファイルをTrainingJobに転送
- entry_point
- source_dir以下のファイル

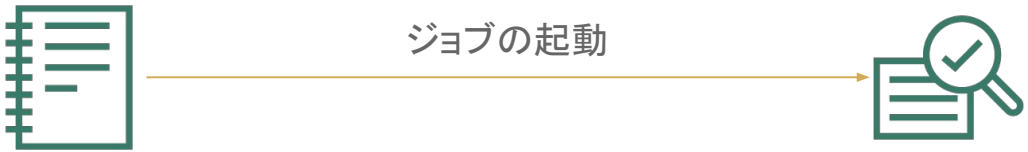
1. requirements.txtがあればinstall
2. entry_pointを実行して学習開始

GPUマシンからSageMakerへの学習環境の移行

- GPUマシンからの移行に必要な作業
  - 学習コードを少し修正（引数の受け取り方などを少し変えるだけ）
  - 学習用データは手動でS3にアップロード
  - Jupyter Notebookを新しく作成（20行程度）

学習コードについて

- ジョブ作成時にentry_point, source_dirを指定する
- entry_pointはSageMaker用に作成 or 修正が必要
- source_dirは基本的にそのまま
  - requirements.txtを追加して必要な環境を構築する



以下のファイルをTrainingJobに転送
- entry_point
- source_dir以下のファイル

1. requirements.txtがあればinstall
2. entry_pointを実行して学習開始

entry_pointについて

- 基本的にはローカルで学習する時のmain関数と同じ
- パラメータの取得やデータの取得・保存場所の修正が必要

```python
import argparse
import pandas as pd

if __name__ == '__main__':
    """
    mainの中に学習の処理を書く
    """
    parser = argparse.ArgumentParser()
    parser.add_argument('--train', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
    parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
    args, _ = parser.parse_known_args()

    # 学習用データのロード
    # args.train 以下に学習用データがある
    training_path = os.path.join(args.train, 'training_data.csv')
    df = pd.read_csv(training_path)

    # 学習する

    # 学習結果を保存する
    # args.model_dir に保存
    clf.save(os.path.join(args.model_dir, 'model.npz'), option='npz')
```

環境変数のリストは
aws/sagemaker-containersのREADMEにある

学習を実行するJupyter Notebook

- 学習を実行するだけなので記述量はこれぐらい
  - sagemaker-python-sdkが入っているので活用すると簡単

```
# モデルの作成・学習の実行
from sagemaker.chainer import Chainer

# パラメータ
train_instance_type = 'ml.p3.2xlarge'
training_data_path = 's3://path_to_training_data.csv'

chainer_estimator = Chainer(entry_point='chainer-train-entry.py',
                            source_dir ="./src",
                            role=role,
                            train_instance_type=train_instance_type,
                            train_instance_count=1,
                            framework_version='5.0.0',
                            hyperparameters={'epochs': 10, 'batch-size': 64}
                            )

chainer_estimator.fit(training_data_path)
```

- 学習コード
- 学習データ
- Training Jobの
  インスタンスタイプ

などを指定して実行

Notebookを便利に使う

- Git リポジトリ連携
  - リポジトリを指定してNotebookインスタンスの起動時に自動でcloneすることができる
  - 起動時に必要なコードが揃っているのでNotebookを実行するだけ
- S3との連携
  - git管理できない大きなファイルはS3に保存してNotebookから取得

Notebook作成時にcloneするリポジトリを指定

```
# 必要なファイルをダウンロード
import boto3

s3 = boto3.resource('s3')
bucket = s3.Bucket('bucket-name')
bucket.download_file('bert.zip', './src/bert.zip')
!cd src && unzip bert.zip
```

大きなファイルはNotebookからダウンロード

SageMakerを使って学習して良かったこと

- 学習に必要な環境を高速に用意できる
  - GPUマシンの時はドライバやライブラリのインストールで躓いた
  - 公式イメージ + 足りないものをrequirements.txtに記述するだけで開発環境の出来上がり
- 必要な環境を必要な数だけ作成することができる
  - ハイパーパラメータの調整・検証のサイクルを早く回せる
  - コンセプト検証段階を終えて本格的に進める際にはスピード感はとても大事

分析やモデル構築など本質的な問題に集中することができる

mixi GROUP

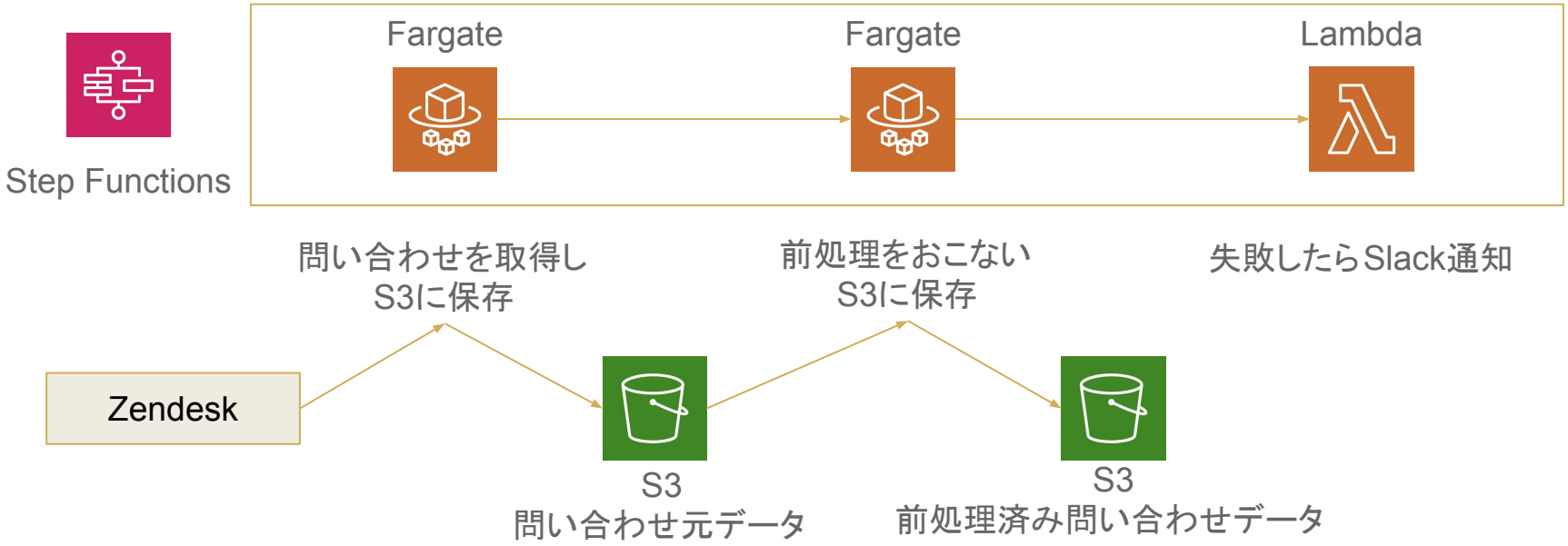
SageMakerと他のサービスとの連携について

SageMakerと他のサービスの連携

- SageMakerは単体でも便利だが、他のサービスと組み合わせると
  より便利に活用することができる
- 今回紹介する事例
  - Step Functions
  - Athena
  - Parameter Store

SageMaker x Step Functions

- データを前処理してS3に保存する処理を作成
- Step FunctionsはSageMakerにも対応しているので、「S3からデータを取得→取得したデータをバッチ推論」といったことも可能



SageMaker x Athena

- S3に保存した前処理済みのデータをAthenaから取得する
- データ分析・アノテーション対象のデータ抽出・バッチ推論用のデータ抽出など様々な場面で活用している

```
import sys
!{sys.executable} -m pip install PyAthena
```

↑
Athenaを使うために
必要なライブラリを入れる

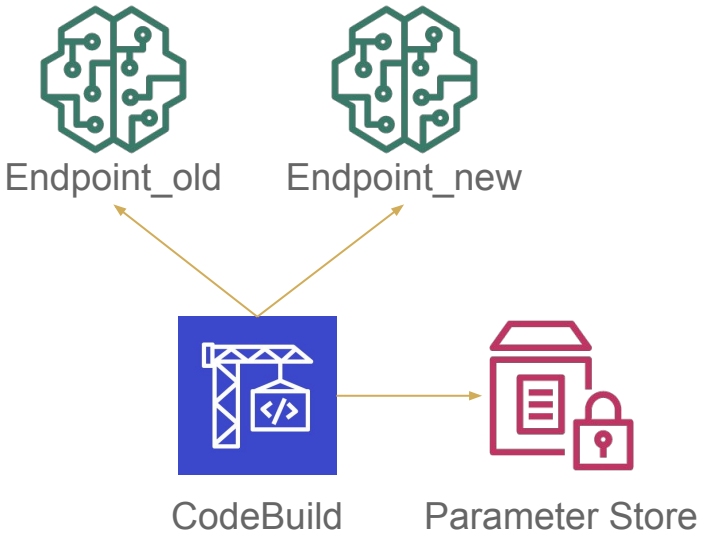
```
from pyathena import connect
import pandas as pd
conn = connect(s3_staging_dir='s3://[redacted]
               region_name='ap-northeast-1')

query = "SELECT * FROM zendesk_inquiry.first_responses \
WHERE (year = '2019' AND month = '09') \
AND inquiry_text LIKE '%クエスト%' \
limit 10;"
df = pd.read_sql(query, conn)
df
```

クエリを実行している様子 →

ticket_id	created_at	inquiry_text
[redacted]		[redacted]のクエストで、マルチでクエストをクリアすれば金卵2個以上ドロップとなってますが先ほ...

SageMaker x Parameter Store

- モデルの更新 = 新しいエンドポイントを立ててそちらを見る
- パラメータストアを参照してエンドポイント名を取得
  - productionとdevelopmentで見る先を変えることも簡単
  - 元のエンドポイントが残っていればロールバックも簡単




```
res = self.sagemaker_runtime.invoke_endpoint(
        EndpointName=endpoint,
        Body=body,
        ContentType='application/json',
        Accept='application/json'
        )
```

SageMakerで推論するコード

まとめ

- GPUマシンからSageMakerへの移行は簡単
  - 少し追加や修正が必要な箇所はあるが簡単に移行できた
- SageMakerを使うことで本質的な問題に集中することができる
  - 学習に必要な環境を高速に用意できる
  - 必要な環境を必要な数だけ作成できる
- SageMakerは他のサービスと組み合わせるとさらに活用できる
  - StepFunctionsを用いたパイプライン構築
  - Athenaを用いて必要なデータをサクッと取得
  - Parameter Storeを用いてエンドポイントを切り替え

mixi
GROUP

ご静聴ありがとうございました！