運用環境でも ハイパフォーマンスな AI

インテル® AI アナリティクス・ツールキット、MLflow*、AWS* を使用した MLOps* 環境の構築

Eduardo Alvarez インテル コーポレーション シニア AI ソリューション・エンジニア

マシンラーニング（ML）アプリケーションを研究開発環境から運用環境に移行する際に、MLOps* は重要なコンポーネントでありながら見落とされがちです。この課題には、通常、ケースバイケースで取り組むのがベストです。この記事では、オープンソースのツールを使ってクラウド上に MLOps* 環境を構築し、ハイパフォーマンスなモデルの開発と展開を確実に行う一般的なブループリントを提示します（**図 1**）。ML コンポーネントのバックボーンとして、インテル® AI アナリティクス・ツールキット、特にインテル® ディストリビューションの Python* と PyTorch* 向けインテル® エクステンションを活用して、トレーニングと推論のパフォーマンスを向上します。この記事を読むことで、モデルの実験、管理、展開を高速に行う仕組みを、サーバーレスのクラウド環境で自由に使えるようになります。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

図 1. 本記事で紹介する MLOps* ブループリントを利用したソリューションのサーバーサイド、運用サイド、クライアント・サイドの各コンポーネントの役割を担う各種ツール。ここでは、Docker*、GitHub*、およびフロントエンド・コンポーネントは取り上げません。

MLOps*—モデルの目標パフォーマンス

MLOps* は、ML と DevOps の俊敏性と回復力を結びつけ、ML アセットを CI/CD パイプラインに関連付けて、運用環境への安定した展開を実現します。これにより、モデルの鮮度とドリフトの懸念に対応した、統一されたリリースプロセスを構築します。適切な MLOps* パイプラインがなければ、ML アプリケーション・エンジニアは高品質の ML アセットを事業部門に提供できず、研究開発部門の中で大きな科学実験を行っているだけになってしまいます。

MLOps* への投資が進まない主な理由は、モデル / データドリフトの影響と、それが運用環境のアプリケーションにどのような影響を与えるかについての理解が不十分であることに起因しています。モデルドリフトとは、データの変化や入力変数と出力変数の関係によって、モデルのパフォーマンスが低下することを指します。データドリフトはモデルドリフトの一種で、独立変数の性質が変化するものです。データドリフトの例として、季節性によるデータの変化、消費者の嗜好の変化、新製品の追加などが挙げられます。確立された MLOps* パイプラインは、データ環境に最適なモデルを展開することで、これらの影響を軽減できます。目標は、運用環境で常に最もパフォーマンスの高いモデルを導入することです。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

MLOps* ソリューション—MLflow*

MLflow* は、オープンソースの ML ライフサイクル管理プラットフォームです（**図 2**）。MLflow* は、クラウド上で動作するあらゆる ML ライブラリーと連携可能で、ビッグデータのワークフローにも容易に対応できる拡張性を備えています。MLflow* は 4 つの部分から構成されています。

- **MLflow* Tracking** は、モデルのトレーニング・プロセスのコード、データ、設定、結果を記録し、照会できるようにします。
- **MLflow* Projects** は、データ・サイエンス・コードをパッケージ化して再展開することで、あらゆるプラットフォームで再現性を確保します。
- **MLflow* Models** は、さまざまな環境にモデルを容易に展開するための API です。
- **MLflow* Model Registry** は、中央のリポジトリーにモデルを保存し、アノテーションを付け、管理できるようにします。

この記事では、チーム全体で利用可能なパスワードで保護された MLflow* サーバーをセットアップする方法を説明します。

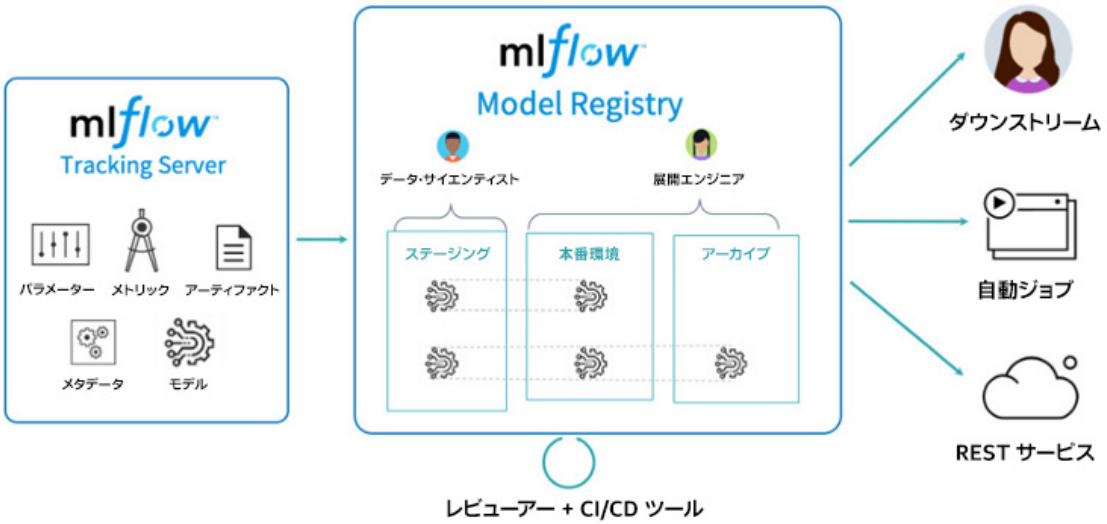


図 2. MLflow* のトラッキングとレジストリー・コンポーネント（出典：Databricks）

MLflow* サーバー—AWS* ソリューション・インフラストラクチャー設計

ここで紹介する MLOps* 環境のクラウド・コンポーネントは単純です。ML モデルをトレーニング、トラッキング、登録、展開するには、インフラストラクチャーが必要です。**図 3** にクラウド・ソリューションを示します。ローカルホスト（または別のリモートホスト）は、さまざまな MLflow* API を使用して、リモートホストのリソースと通信します。リモートホストは、トラッキング・サーバーをホストし、データベースやオブジェクト・ストレージと通信する役割を担います。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

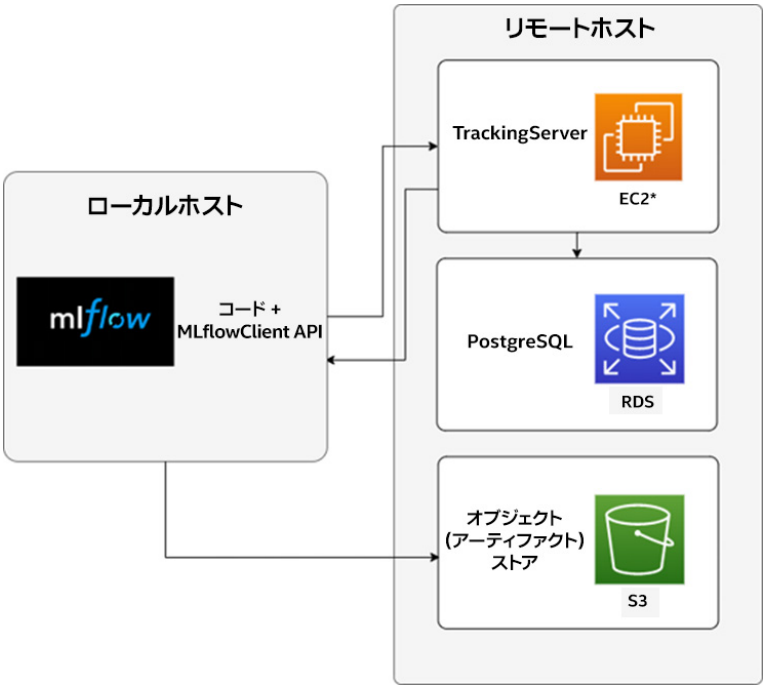


図 3. リモートホストとローカルホストのコンポーネントを示す AWS* ソリューション・アーキテクチャー

このアーキテクチャーをセットアップする手順は後述します。ここでは、AWS* の無料利用枠を利用しますが、これは運用環境の構築には推奨しません。

表 1 は、製品デモのセットアップに使用したリソースを示しています。このブループリントをベースに、ニーズに応じて調整できます。選択したインスタンスはすべて第 3 世代インテル® Xeon® スケーラブル・プロセッサーで、vCPU と GiB 容量が異なります。

AWS* サービス	用途	リソース要件
EC2*	ホスト MLFlow サーバー	m6i.xlarge (4 vCPU、16GiB メモリー)、8GB ブロックストレージ
EC2*	ホスト Jupyter* サーバー	m6i.4xlarge (16 vCPU、64GiB メモリー)、256GB ブロックストレージ
RDS	MLFlow バックエンド・ストア	任意の適切な DB インスタンス、100GB 以上のストレージ
S3	MLFlow オブジェクト・ストア	デフォルトの設定 (必要に応じて、バックアップ頻度を設定)

表 1. 汎用環境でテストされた、ブループリントの各種コンポーネントの AWS* サービスとリソース要件

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

マシンラーニング・フレームワーク—インテル® AI アナリティクス・ツールキット

MLOps* 環境では、モデルを構築するため、ハイパフォーマンスの ML フレームワークのセットが必要です。インテル® AI アナリティクス・ツールキットは、低レベルの計算を最適化するため、oneAPI ライブラリーを使用して構築された Python* ツールおよびフレームワークを提供します。このツールキットを活用することで、インテル® XPU でハイパフォーマンスなディープラーニング・トレーニングを提供し、高速な推論をワークフローに統合できる、MLOps* 環境を実現できます。

表 2 は、エンドツーエンドの ML ライフサイクルと、MLOps* ブループリントにおける各コンポーネントの説明です。インテル® AI アナリティクス・ツールキットがパフォーマンスの向上と計算コストの軽減を支援しているコンポーネントは、太字で強調表示されています。

ライフ・サイクル・コンポーネント	ソリューション	説明
データストア	クラウド	データレイクやデータベースへのデータの取り込み
データサイエンスの実験	**インテル® AI アナリティクス・ツールキット**	ML モデルの研究開発 (R&D)
バージョン管理	GitHub*	コード管理
DevOps	GitHub*	CI/CD ステージ: ビルド、テスト、パッケージ化、展開パイプライン
特徴エンジニアリング	**インテル® AI アナリティクス・ツールキット**	モデルの文脈を改善するデータ変換
ML モデル・エンジニアリング	**インテル® AI アナリティクス・ツールキット**	モデルのチューニング、転移学習など
モデルの登録	MLOps* フレームワーク (MLflow* またはその他)	運用環境用モデルの格納
モデルの供給	MLOps* フレームワーク (MLflow* またはその他)	運用環境用モデルをエンドポイントへ供給
ML 予測サービス	**OpenVINO™ ツールキット**	モデル予測 API へデータを供給
パフォーマンス・モニタリング	MLOps* フレームワーク (MLflow* またはその他)	ライブデータのモデル・パフォーマンスに関するデータを収集
パフォーマンス・ベースのトリガー	MLOps* フレームワーク (MLflow* またはその他)	モデル・パフォーマンスによって追加のワークフローをトリガー

表 2. エンドツーエンドの ML ライフサイクル・ソリューション

conda 環境のセットアップは本題から外れますが、参考までに以下に環境設定スクリプト（`aikit_ipex.yml`）を示します。次のコマンドで適切な conda 環境を作成できます。

```
conda env create -f aikit_ipex.yml
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

```yaml
name: aikit_ipex
channels:
  - conda-forge
  - intel
  - defaults
dependencies:
  - cpuonly
  - intelpython
  - intelpython3_core
  - ipykernel
  - jupyterlab
  - numpy
  - pandas
  - pip
  - python
  - scikit-learn
  - scipy
  - pip:
    - boto3==1.24.59
    - mlflow==1.28.0
    - seaborn==0.11.2
    - torch==1.12.1
    - torchvision==0.13.1
```

aikit_ipex.yml hosted with ❤ by GitHub view raw

MLflow* リモートホストのセットアップ

AWS* EC2* を使用したホストマシンのセットアップ

1. AWS* マネジメント・コンソールにアクセスし、新しい EC2* インスタンスを起動します。ここでは Amazon* Linux* を選択していますが、リストからほかの Linux* や Windows* を使用することもできます。
2. キーペアがない場合は、新規に作成します。キーペアは、SSH でインスタンスにアクセスし、サーバーの各種設定を行うのに必要です。
3. 新しいセキュリティー・グループを作成し、任意の場所（0.0.0.0/0）からの SSH、HTTPS、HTTP トラフィックを許可します。ただし、最大限のセキュリティーを確保するため、インターネット全体に接続を開放するのではなく、サーバーにアクセスできる特定の IP をセーフリストに登録することを推奨します。
4. サーバーの EC2* ストレージボリュームは、このインスタンスをほかの用途に使用する予定がない限り、8 ～ 16GB より大きくする必要はありません。このインスタンスは、MLflow* 専用サーバーにすることを推奨します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

S3 オブジェクト・ストアのセットアップ

S3 バケットは、モデルのアーティファクト、環境設定、コードバージョン、データバージョンの保存に使用できます。ここに、MLflow* ML 管理パイプラインの重要な部分がすべて含まれています。

1. AWS* マネジメント・コンソールから新しいバケットを作成します。
2. ACL を有効にし、すべてのパブリックアクセスをブロックしたままにします。ACL を有効にすることで、適切な認証情報を持つほかの AWS* アカウントからバケット内のオブジェクトにアクセスできるようになります。

AWS* RDS の PostgreSQL データベースのセットアップ

MLflow* ワークフローのこのコンポーネントは、実行、パラメーター、メトリック、タグ、ノート、オブジェクト・ストアへのパス、およびその他のメタデータを保存します。

1. AWS* マネジメント・コンソールからデータベースを作成します。今回は PostgreSQL データベースを使用します。無料利用枠を選択します。
2. データベースに名前を付け、マスターユーザー名を割り当て、このアカウントのパスワードを指定します。この情報は、MLflow* サーバーを起動する際に使用します。
3. インスタンスの種類を選択します。MLflow* サーバーを経由する予定のトラフィック量（つまり、モデルやデータのクエリー回数）に応じて、より強力なインスタンスが必要になる場合があります。
4. ストレージも指定する必要があります。これもトラッキングする必要があるメタデータの量に依存します。モデルやアーティファクトはここに保存されないので、大きなスペースは必要ないでしょう。
5. 仮想プライベート・クラスターの外からデータベースの読み書きができるように、パブリックアクセスは必須です。次に、セキュリティー・グループを使用して、セーフリスト IP を指定します。
6. 任意の場所（0.0.0.0/0）または特定の IP からのすべての TCP トラフィックを許可する受信規則を持つセキュリティー・グループを作成する必要があります。
7. RDS を起動します。

AWS* CLI をインストールして SSH でインスタンスにアクセスする

AWS* CLI は、コマンドラインから AWS* リソースを制御するツールです。こちらのリンク（英語）にアクセスして、AWS* ウェブサイトから CLI をインストールできます。AWS* CLI をインストールすると、S3 オブジェクト・ストアへの書き込みを許可する AWS* 認証情報を自分のマシンに設定できます。これは、MLflow* をスクリプト、ノートブック、API に組込む際に必要になります。

1. EC2* インスタンスのダッシュボードから、MLflow* EC2* インスタンスを選択し、**[Connect]** をクリックします。
2. **[SSH]** クライアント・タブに移動し、プロンプトの下部からクリップボードに SSH の例をコピーします。
3. プロンプトに SSH コマンドを貼り付けます。ここでは、.pem ファイルを保存したフォルダーから Git Bash を使用しています。
4. 以下の依存関係をインストールします。

```
sudo pip3 install mlflow[extras] psycopg2-binary boto3

sudo yum install httpd-tools

sudo amazon-Linux-extras install nginx1
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

5. nginx ユーザーを作成して、パスワードを設定します。

```
sudo htpasswd -c /etc/nginx/.htpasswd testuser
```

6. 最後に、nginx リバースプロキシーをポート 5000 に設定します。

```
sudo nano /etc/nginx/nginx.conf
```

nano を使って、nginx の設定ファイルを編集し、リバースプロキシーに関する以下の情報を追加します。

```
location / {
proxy_ass http://localhost:5000/;
auth_basic “Restricted Content”;
auth_basic_user_file /etc/nginx/.htpasswd;
}
```

add2nginx.txt hosted with ❤ by GitHub view raw

最終的なスクリプトは、**図 4** に示すコンソールの nano 出力のようになるはずです。

```
GNU nano 2.9.8                                  /etc/nginx/nginx.conf

keepalive_timeout   65;
types_hash_max_size 4096;

include             /etc/nginx/mime.types;
default_type        application/octet-stream;

# Load modular configuration files from the /etc/nginx/conf.d directory.
# See http://nginx.org/en/docs/ngx_core_module.html#include
# for more information.
include /etc/nginx/conf.d/*.conf;

server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /usr/share/nginx/html;

    location / {
            proxy_pass http://localhost:5000/;
            auth_basic "Restricted Content";
            auth_basic_user_file /etc/nginx/.htpasswd;
    }

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;
```

図 4. サーバーの場所を追加した後のプロキシーサーバー nginx.conf の nano 出力

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

nginx と MLflow* サーバーの起動

nginx リバース・プロキシー・サーバーを起動し、MLflow* のリモートサーバーを実行できるようになりました。この手順では、AWS* マネジメント・コンソールからいくつかの情報を取得する必要があります。

1. nginx サーバーを起動します。

   ```
   sudo service nginx start
   ```

2. MLflow* サーバーを起動し、オブジェクト・ストレージ（S3）とバックエンド・ストレージ（RDS）のすべてのコンポーネントを設定します。

   ```
   mlflow server --backend-store-uri
   postgresql://MASTERUSERNAME:YOURPASSWORD@YOUR-DATABASE-
   ENDPOINT:5432/postgres --default-artifact-root s3://BUCKETNAME --
   host 0.0.0.0
   ```

この情報はどこにあるのでしょうか？

- MASTERUSERNAME—PostgreSQL RDS DB に設定したユーザー名です。
- YOURPASSWORD—PostgreSQL RDS DB に設定したパスワードです。
- YOUR-DATABASE-ENDPOINT—これは、AWS* マネジメント・コンソール内の RDS DB 情報で見つけることができます。
- BUCKETNAME—S3 バケットの名前です。

このコマンドを実行すると、大量の出力が表示され、ワーカー ID と pid に関する情報が得られます。

```
INFO  [alembic.runtime.migration] Running upgrade 181f10493468 -> df50e92ffc5e, Add Experiment Tags Table
INFO  [alembic.runtime.migration] Running upgrade df50e92ffc5e -> 7ac759974ad8, Update run tags with larger limit
INFO  [alembic.runtime.migration] Running upgrade 7ac759974ad8 -> 89d4b8295536, create latest metrics table
INFO  [89d4b8295536_create_latest_metrics_table_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 89d4b8295536 -> 2b4d017a5e9b, add model registry tables to db
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Adding registered_models and model_versions tables to database.
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 2b4d017a5e9b -> cfd24bdc0731, Update run status constraint with killed
INFO  [alembic.runtime.migration] Running upgrade cfd24bdc0731 -> 0a8213491aaa, drop_duplicate_killed_constraint
INFO  [alembic.runtime.migration] Running upgrade 0a8213491aaa -> 728d730b5ebd, add registered model tags table
INFO  [alembic.runtime.migration] Running upgrade 728d730b5ebd -> 27a6a02d2cf1, add model version tags table
INFO  [alembic.runtime.migration] Running upgrade 27a6a02d2cf1 -> 84291f40a231, add run_link to model_version
INFO  [alembic.runtime.migration] Running upgrade 84291f40a231 -> a8c4a736bde6, allow nulls for run_id
INFO  [alembic.runtime.migration] Running upgrade a8c4a736bde6 -> 39d1c3be5f05, add_is_nan_constraint_for_metrics_tables
_if_necessary
INFO  [alembic.runtime.migration] Running upgrade 39d1c3be5f05 -> c48cb773bb87, reset_default_value_for_is_nan_in_metric
s_table_for_mysql
INFO  [alembic.runtime.migration] Running upgrade c48cb773bb87 -> bd07f7e963c5, create index on run_uuid
INFO  [alembic.runtime.migration] Running upgrade bd07f7e963c5 -> 0c779009ac13, add deleted_time field to runs table
INFO  [alembic.runtime.migration] Running upgrade 0c779009ac13 -> cc1f77228345, change param value length to 500
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
[2022-09-05 17:16:28 +0000] [6701] [INFO] Starting gunicorn 20.1.0
[2022-09-05 17:16:28 +0000] [6701] [INFO] Listening at: http://0.0.0.0:5000 (6701)
[2022-09-05 17:16:28 +0000] [6701] [INFO] Using worker: sync
[2022-09-05 17:16:28 +0000] [6704] [INFO] Booting worker with pid: 6704
[2022-09-05 17:16:28 +0000] [6705] [INFO] Booting worker with pid: 6705
[2022-09-05 17:16:28 +0000] [6706] [INFO] Booting worker with pid: 6706
[2022-09-05 17:16:28 +0000] [6707] [INFO] Booting worker with pid: 6707
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

MLflow* サーバーのシャットダウンまたはバックグラウンドで実行

MLflow* サーバーをシャットダウンする場合は、以下のコマンドを使用します。

```
sudo fuser -k <port>/tcp
```

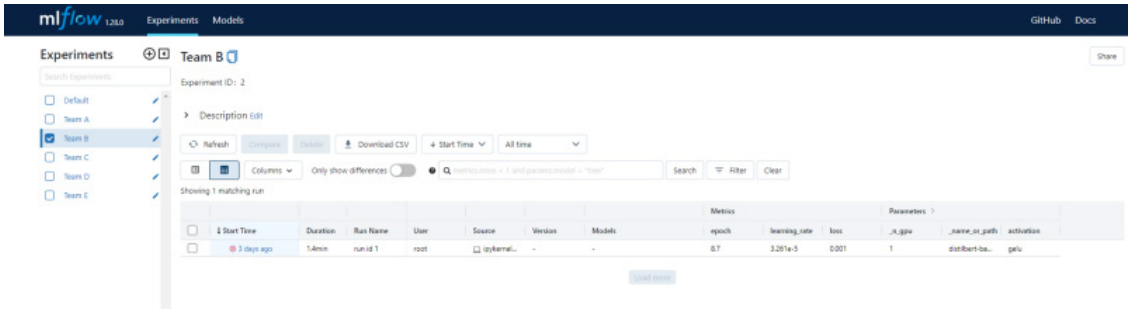
ターミナルを閉じてバックグラウンドでサーバーの実行を継続する場合は、MLflow* サーバーの起動コマンドに `nohup` と `&` を追加します。

```
nohup mlflow server --backend-store-uri
postgresql://MASTERUSERNAME:YOURPASSWORD@YOUR-DATABASE-
ENDPOINT:5432/postgres --default-artifact-root s3://BUCKETNAME --
host 0.0.0.0 &
```

ブラウザーから MLflow* UI にアクセスする

MLflow* サーバーのインフラストラクチャーのセットアップが完了したので、コマンドラインやスクリプトから、MLflow* サーバーを利用できます。MLflow* の UI にアクセスするには、EC2* インスタンスの IPV4 パブリック IP アドレスが必要です。

1. IPV4 パブリック IP をコピー & ペーストします (これは、EC2* マネジメント・コンソール内で見つけることができます)。IP アドレスの前に、http:// があることを確認します。
2. nginx のユーザー名とパスワードを入力するプロンプトが表示されます。
3. これで、MLflow* の UI から実験やモデルにアクセスし、管理できるようになりました。



コードへの MLflow* の追加とインテル ® AI リファレンス・キットの例

MLflow* トラッキング・サーバー

まず、開発環境内で MLflow* の認証情報を追加し、トラッキング URI を設定する必要があります。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

```python
import mlflow

# setting up tracking server uri
# this is your remote server EC2 instance IPV4 Public IP
tracking_uri = ''
client = mlflow.tracking.MlflowClient(tracking_uri=tracking_uri)

# nginx credentials
# these are the same credentials that you created during your nginx config step
os.environ['MLFLOW_TRACKING_USERNAME'] = 'testuser'
os.environ['MLFLOW_TRACKING_PASSWORD'] = 'password'
os.environ['MLFLOW_TRACKING_URI'] = tracking_uri
```

mlflow_config.py hosted with ❤ by GitHub view raw

VGG16 モデルを使用して画像を分類するモデルをトレーニングしてみましょう（残りのソースコードは、インテル® AI リファレンス・キットのビジュアル品質検査リポジトリー（英語）を参照してください）。

```python
# mlflow training stuff
experiment_name = 'Team A'
mlflow.set_experiment(experiment_name) # creation of an new experiment
mlflow_artifact_path='artifacts' # this is the folder name within your S3 bucket where model will
run_id = 'run 1' # name for current mlflow training run


# initializing an mlflow run
with mlflow.start_run(run_name=run_id) as run:

    # Intitalization of DL architecture along with optimizer and loss function
    model = CustomVGG()
    class_weight = torch.tensor(class_weight).type(torch.FloatTensor).to(DEVICE)
    criterion = nn.CrossEntropyLoss(weight=class_weight)
    optimizer = optim.Adam(model.parameters(), lr=LR)

    # intel extension for PyTorch Optimization
    model, optimizer = ipex.optimize(model=model, optimizer=optimizer, dtype=torch.float32)


    # Training section
    start_time = time.time()
    trained_model = train(data_loader=train_loader, model=model, optimizer=optimizer, criterion=cr
        device=DEVICE, target_accuracy=TARGET_TRAINING_ACCURACY, data_aug=data_aug)
    train_time = time.time()-start_time

    mlflow.pytorch.log_model(pytorch_model=trained_model, artifact_path=mlflow_artifact_path)
    mlflow.log_metrics('Training Time', train_time)

    latest_run_id = run.info.run_id

# ending the mlflow run
mlflow.end_run()
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

MLflow* モデル・レジストリー

ML の実験を行い、運用環境に投入する価値のあるいくつかのモデルを定義したら、それらを登録し、展開するためステージングできます。以下のブロックでは、モデルの実行 ID と実験名を使用して、MLflow* モデル・レジストリーにモデルを追加しています。そして、このモデルを「運用」環境に移動してロードし、異常な錠剤を特定する予測サービスとして使用しています。

```
import mlflow.pyfunc

# adding a model to the model registry
model_uri=f"runs:/{current_run_id}/{mlflow_artifact_path}"
name='Team A'
await_registration_for=5
mlflow.register_model(model_uri, name, await_registration_for)

# move a model to archived, staging, and production
client.transition_model_version_stage(name="Team A", version=8, stage="Production")

# Loading latest production version of a model
model_name = 'Team A'
stage = 'Production'
model_uri= f"models:/{model_name}/{stage}"
mlflow_model = mlflow.pytorch.load_model(model_uri)

# predicting with latest production version of a model
predict_localize(mlflow_model, test_loader, DEVICE, thres=HEATMAP_THRESH, n_samples=3, show_heatma
```

mlflow_model_register&serving.py hosted with ❤ by GitHub view raw

図 5 から、運用環境に展開したモデルが非常にうまく機能していることが分かります。

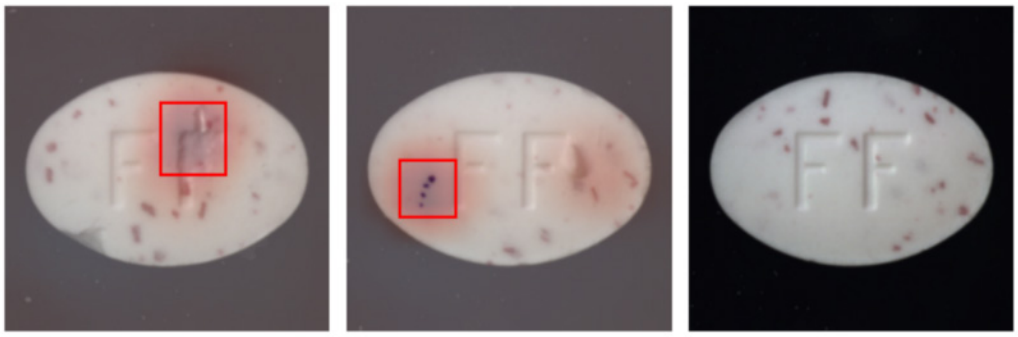


図 5. 異常な錠剤（左と中央）と正常な錠剤（右）を識別するために展開されたモデル

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

まとめ

この記事では、インテル® AI アナリティクス・ツールキットを利用して、高速なエンドツーエンドの ML パイプラインを構築する、一般的なクラウドベースの MLOps* 環境を紹介しました。このパイプラインは、運用環境で常に最もパフォーマンスの高いモデルを提供するための重要な第一歩です。この記事の目的は、ありふれた環境を提案することではありません。パイプラインを実装するには、計算、セキュリティー、および一般的なアプリケーションの要件を考慮し、どこに変更を加える必要があるかを認識することが不可欠です。

intel. 1 oneAPI BASE TOOLKIT

インテル® oneAPI ベース・ツールキットは、さまざまなアーキテクチャーに対応したハイパフォーマンスなアプリケーションやソリューションの開発を可能にします。

詳細

『oneAPI GPU 最適化ガイド』を入手して、並列化からカーネル、メモリーに至るまで最高のパフォーマンスを実現してください。

詳細

Github* にアクセスして oneAPI サンプルコードを取得しましょう。

Github* にアクセス（英語）

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。