# NetAppによるオープンソース MLOps

NetApp artificial intelligence solutions

NetApp
August 18, 2025

# 目次


NetAppによるオープンソース MLOps 1
NetAppによるオープンソース MLOps 1
テクノロジの概要 2
人工知能 3
コンテナ 3
Kubernetes 3
NetAppTrident 4
NetApp DataOps ツールキット 4
Apacheエアフロー 4
Jupyterノートブック 4
ジュピターハブ 5
MLフロー 5
キューブフロー 5
NetApp ONTAP 6
NetAppスナップショットコピー 7
NetApp FlexCloneテクノロジ 8
NetApp SnapMirrorデータレプリケーションテクノロジー 9
NetApp BlueXPコピーと同期 9
NetApp XCP 10
NetApp ONTAP FlexGroupボリューム 10
アーキテクチャ 10
Apache Airflow 検証環境 11
JupyterHub 検証環境 11
MLflow 検証環境 11
Kubeflow 検証環境 11
サポート 12
NetApp Trident の構成 12
NetApp AIPod導入におけるTridentバックエンドの例 12
NetApp AIPodデプロイメント用の Kubernetes ストレージクラスの例 14
Apacheエアフロー 17
Apache Airflow デプロイメント 17
NetApp DataOps Toolkit を Airflow と併用する 21
ジュピターハブ 21
JupyterHub デプロイメント 21
JupyterHubでNetApp DataOpsツールキットを使用する 24
NetApp SnapMirrorを使用してJupyterHubにデータを取り込む 27
MLフロー 27
MLflow デプロイメント 27
NetAppと MLflow によるデータセットからモデルへのトレーサビリティ 29

キューブフロー . . . 30
Kubeflow デプロイメント . . . 30
データ サイエンティストまたは開発者向けに Jupyter Notebook ワークスペースをプロビジョニングする . . . 31
KubeflowでNetApp DataOpsツールキットを使用する . . . 32
ワークフロー例 - Kubeflow とNetApp DataOps Toolkit を使用して画像認識モデルをトレーニングする . . . 32
Trident操作の例 . . . 35
既存のボリュームをインポートする . . . 35
新しいボリュームをプロビジョニングする . . . 37
AIPod展開の高パフォーマンスジョブの例 . . . 38
単一ノードのAIワークロードを実行する . . . 38
同期分散AIワークロードを実行する . . . 42

# NetAppによるオープンソース MLOps

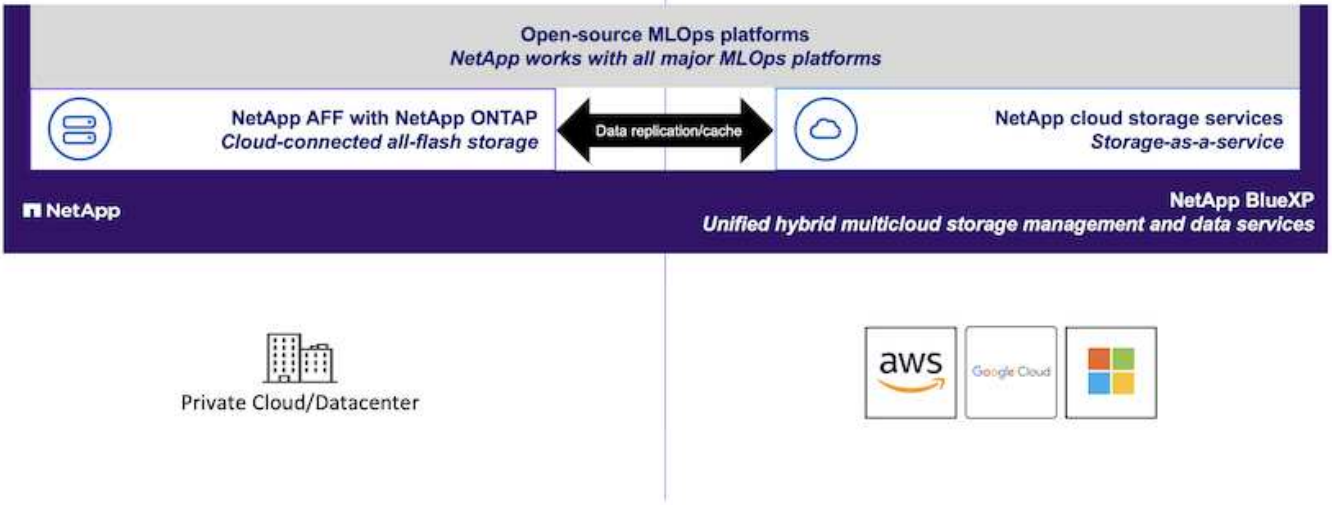
## NetAppによるオープンソース MLOps

Mike Oglesby、 NetApp Sufian Ahmad、 NetApp Rick Huang、 NetApp Mohan Acharya、 NetApp

あらゆる規模や業種の企業や組織が、現実世界の問題を解決し、革新的な製品やサービスを提供して、競争が激化する市場で優位に立つために、人工知能 (AI) を導入しています。多くの組織は、業界の急速なイノベーションのペースに対応するために、オープンソースの MLOps ツールに注目しています。これらのオープンソース ツールは高度な機能と最先端の機能を提供しますが、データの可用性とデータのセキュリティが考慮されていないことがよくあります。残念ながら、これは高度なスキルを持つデータ サイエンティストが、データにアクセスできるようになるまで、または基本的なデータ関連の操作が完了するまで、かなりの時間を費やさざるを得ないことを意味します。人気のオープンソース MLOps ツールとNetAppのインテリジェント データ インフラストラクチャを組み合わせることで、組織はデータ パイプラインを高速化し、AI イニシアチブを加速できます。データの保護とセキュリティを維持しながら、データから価値を引き出すことができます。このソリューションは、これらの課題に対処するために、 NetApp のデータ管理機能といくつかの一般的なオープンソース ツールおよびフレームワークを組み合わせる方法を示しています。

次のリストは、このソリューションによって有効になる主な機能の一部を示しています。

- ユーザーは、高性能なスケールアウトNetAppストレージを活用した新しい大容量データ ボリュームと開発ワークスペースを迅速にプロビジョニングできます。
- ユーザーは、実験や迅速な反復を可能にするために、大容量のデータ ボリュームと開発ワークスペースをほぼ瞬時に複製できます。
- ユーザーは、バックアップやトレーサビリティ/ベースライン作成のために、大容量データ ボリュームや開発ワークスペースのスナップショットをほぼ瞬時に保存できます。




典型的なMLOpsワークフローには開発ワークスペースが組み込まれており、通常は次のような形式をとります。"Jupyterノートブック" ; 実験の追跡、自動トレーニング パイプライン、データ パイプライン、および推論/デプロイメント。このソリューションは、ワークフローのさまざまな側面に対処するために、独立して、または組み合わせて使用できるさまざまなツールとフレームワークに重点を置いています。また、 NetApp のデータ管理機能とこれらの各ツールの組み合わせについても説明します。このソリューションは、組織がユースケースと要件に合わせてカスタマイズされた MLOps ワークフローを構築するための構成要素を提供することを目的としています。

このソリューションでは、次のツール/フレームワークがカバーされています。

- "Apacheエアフロー"
- "ジュピターハブ"
- "キューブフロー"
- "MLフロー"

次のリストでは、これらのツールを個別に、または組み合わせて展開するための一般的なパターンについて説明します。

- JupyterHub、MLflow、Apache Airflowを組み合わせてデプロイ - JupyterHub for"Jupyterノートブック" 、実験追跡用の MLflow、自動トレーニングおよびデータ パイプライン用の Apache Airflow です。
- KubeflowとApache Airflowを組み合わせてデプロイ - Kubeflow for"Jupyterノートブック" 、実験の追跡、自動化されたトレーニング パイプライン、推論、およびデータ パイプライン用の Apache Airflow です。
- KubeflowをオールインワンのMLOpsプラットフォームソリューションとして導入"Jupyterノートブック"、実験の追跡、自動化されたトレーニングとデータ パイプライン、および推論。

# テクノロジの概要

このセクションでは、 NetAppを使用した OpenSource MLOps のテクノロジの概要に焦点を当てます。

## 人工知能

AI は、コンピューターが人間の心の認知機能を模倣するようにトレーニングするコンピューター サイエンスの分野です。 AI 開発者は、コンピューターが人間と同様、あるいは人間よりも優れた方法で学習し、問題を解決できるようにトレーニングします。ディープラーニングと機械学習は AI のサブフィールドです。組織は、重要なビジネスニーズをサポートするために AI、ML、DL を導入するケースが増えています。次にいくつかの例を示します。

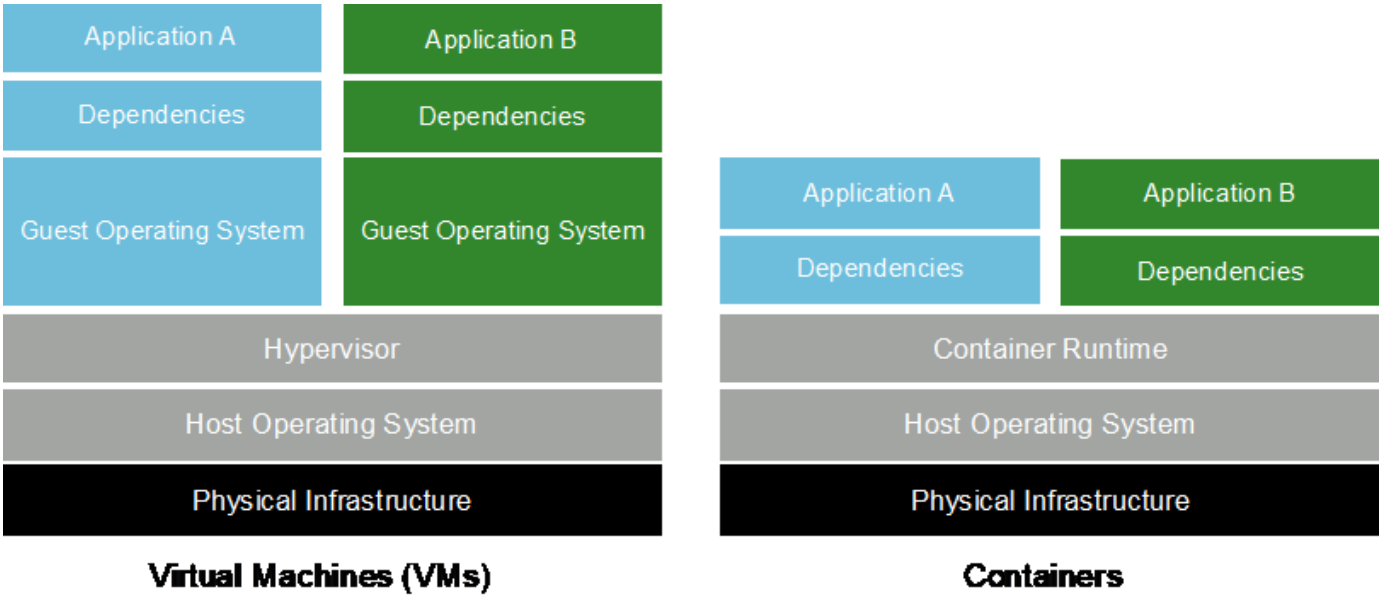
- 膨大なデータを分析し、これまで知られていなかったビジネスインサイトを発見する
- 自然言語処理を使用して顧客と直接対話する
- さまざまなビジネスプロセスと機能の自動化

最新の AI トレーニングおよび推論ワークロードには、超並列コンピューティング機能が必要です。そのため、GPU の並列処理能力は汎用 CPU よりもはるかに優れているため、AI 演算の実行に GPU がますます使用されるようになっています。

## コンテナ

コンテナは、共有ホスト オペレーティング システム カーネル上で実行される分離されたユーザー空間インスタンスです。コンテナの導入が急速に増加しています。コンテナーは、仮想マシン (VM) が提供するのと同じアプリケーション サンドボックスの利点の多くを提供します。ただし、VM が依存するハイパーバイザーとゲスト オペレーティング システム レイヤーが削除されているため、コンテナーははるかに軽量です。次の図は、仮想マシンとコンテナを視覚的に表したものです。

コンテナーを使用すると、アプリケーションの依存関係や実行時間などをアプリケーションに直接効率的にパッケージ化することもできます。最も一般的に使用されるコンテナ パッケージ形式は Docker コンテナです。Docker コンテナ形式でコンテナ化されたアプリケーションは、Docker コンテナを実行できる任意のマシンで実行できます。すべての依存関係はコンテナー自体にパッケージ化されるため、アプリケーションの依存関係がマシン上に存在しない場合でも、これは当てはまります。詳細については、 "Dockerウェブサイト" 。




## Kubernetes

Kubernetes は、もともと Google によって設計され、現在は Cloud Native Computing Foundation (CNCF) に

よって管理されているオープンソースの分散型コンテナ オーケストレーション プラットフォームです。 Kubernetes を使用すると、コンテナ化されたアプリケーションの展開、管理、スケーリング機能を自動化できます。近年、Kubernetes は主要なコンテナ オーケストレーション プラットフォームとして登場しました。詳細については、 "Kubernetesウェブサイト" 。

## NetAppTrident

"Trident"ONTAP (AFF、 FAS、Select、Cloud、 Amazon FSx ONTAP)、 Azure NetApp Filesサービス、 Google Cloud NetApp Volumesなど、パブリック クラウドまたはオンプレミスのすべての一般的なNetAppストレージ プラットフォームにわたるストレージ リソースの使用と管理を可能にします。 Trident は、Kubernetes とネイティブに統合される、Container Storage Interface (CSI) 準拠の動的ストレージ オーケストレーターです。

## NetApp DataOps ツールキット

その"NetApp DataOps ツールキット"は、高性能なスケールアウトNetAppストレージを基盤とする開発/トレーニング ワークスペースと推論サーバーの管理を簡素化する Python ベースのツールです。主な機能は次のとおりです。

- 高性能のスケールアウトNetAppストレージを活用した新しい大容量ワークスペースを迅速にプロビジョニングします。
- 実験や迅速な反復を可能にするために、大容量のワークスペースをほぼ瞬時に複製します。
- バックアップやトレーサビリティ/ベースライン作成のために、大容量のワークスペースのスナップショットをほぼ瞬時に保存します。
- 大容量、高パフォーマンスのデータ ボリュームをほぼ瞬時にプロビジョニング、クローン作成、スナップショット作成します。

## Apacheエアフロー

Apache Airflow は、複雑なエンタープライズ ワークフローのプログラムによる作成、スケジュール設定、監視を可能にするオープン ソースのワークフロー管理プラットフォームです。これは、ETL およびデータ パイプライン ワークフローを自動化するためによく使用されますが、これらのタイプのワークフローに限定されません。 Airflow プロジェクトは Airbnb によって開始されましたが、その後業界で非常に人気となり、現在は Apache Software Foundation の管轄下にあります。 Airflow は Python で記述され、Airflow ワークフローは Python スクリプトを介して作成され、Airflow は「コードとしての構成」の原則に基づいて設計されています。現在、多くのエンタープライズ Airflow ユーザーは Kubernetes 上で Airflow を実行しています。

有向非巡回グラフ（**DAG**）

Airflow では、ワークフローは有向非巡回グラフ (DAG) と呼ばれます。 DAG は、DAG 定義に応じて、順番に、並列に、またはその 2 つの組み合わせで実行されるタスクで構成されます。 Airflow スケジューラは、DAG 定義で指定されたタスク レベルの依存関係に従って、ワーカーの配列で個々のタスクを実行します。 DAG は Python スクリプトによって定義および作成されます。

## Jupyterノートブック

Jupyter Notebook は、ライブ コードと説明テキストを含む wiki のようなドキュメントです。 Jupyter Notebook は、AI および ML プロジェクトを文書化、保存、共有する手段として、AI および ML コミュニティで広く使用されています。 Jupyter Notebookの詳細については、 "Jupyterのウェブサイト" 。

#### Jupyter ノートブック サーバー

Jupyter Notebook サーバーは、ユーザーが Jupyter Notebook を作成できるようにするオープン ソースの Web アプリケーションです。

### ジュピターハブ

JupyterHub は、個々のユーザーが独自の Jupyter Notebook サーバーをプロビジョニングしてアクセスできるようにするマルチユーザー アプリケーションです。 JupyterHubの詳細については、 "JupyterHubウェブサイト" 。

## MLフロー

MLflow は、人気の高いオープンソースの AI ライフサイクル管理プラットフォームです。 MLflow の主な機能には、AI/ML 実験追跡と AI/ML モデル リポジトリが含まれます。 MLflowの詳細については、 "MLflowウェブサイト" 。

### キューブフロー

Kubeflow は、もともと Google によって開発された、Kubernetes 用のオープンソース AI および ML ツールキットです。 Kubeflow プロジェクトは、Kubernetes 上での AI および ML ワークフローのデプロイメントをシンプル、移植可能、かつスケーラブルなものにします。 Kubeflow は Kubernetes の複雑な部分を抽象化し、データ サイエンティストが最も得意とする分野、つまりデータ サイエンスに集中できるようにします。視覚的に表すと次の図を参照してください。 Kubeflow は、オールインワンの MLOps プラットフォームを好む組織にとって優れたオープンソース オプションです。詳細については、 "Kubeflowウェブサイト" 。

#### Kubeflow パイプライン

Kubeflow Pipelines は Kubeflow の重要なコンポーネントです。 Kubeflow Pipelines は、移植可能でスケーラブルな AI および ML ワークフローを定義およびデプロイするためのプラットフォームおよび標準です。詳細については、 "Kubeflowの公式ドキュメント" 。

#### Kubeflow ノートブック

Kubeflow は、Kubernetes 上の Jupyter Notebook サーバーのプロビジョニングとデプロイメントを簡素化します。 Kubeflowのコンテキストにおける Jupyter Notebookの詳細については、 "Kubeflowの公式ドキュメント" 。

#### カティブ

Katib は、自動機械学習 (AutoML) のための Kubernetes ネイティブ プロジェクトです。 Katib は、ハイパーパラメータ調整、早期停止、ニューラル アーキテクチャ検索 (NAS) をサポートしています。 Katib は、機械学習 (ML) フレームワークに依存しないプロジェクトです。ユーザーが選択した任意の言語で記述されたアプリケーションのハイパーパラメータを調整でき、TensorFlow、MXNet、PyTorch、XGBoost などの多くの ML フレームワークをネイティブにサポートします。 Katib は、ベイズ最適化、パルゼン木推定量、ランダム検索、共分散行列適応進化戦略、ハイパーバンド、効率的なニューラル アーキテクチャ検索、微分可能アーキテクチャ検索など、さまざまな AutoML アルゴリズムをサポートしています。 Kubeflowのコンテキストにおける Jupyter Notebookの詳細については、 "Kubeflowの公式ドキュメント" 。

## NetApp ONTAP

NetAppの最新世代のストレージ管理ソフトウェアであるONTAP 9 により、企業はインフラストラクチャを最新化し、クラウド対応のデータセンターに移行できるようになります。 ONTAP は業界をリードするデータ管理機能を活用し、データの保存場所に関係なく、単一のツール セットでデータの管理と保護を可能にします。また、エッジ、コア、クラウドなど、必要な場所にデータを自由に移動することもできます。 ONTAP 9 には、データ管理を簡素化し、重要なデータを高速化および保護し、ハイブリッド クラウド アーキテクチャ全体で次世代のインフラストラクチャ機能を実現する多数の機能が含まれています。

### データ管理を簡素化

データ管理は、AI アプリケーションと AI/ML データセットのトレーニングに適切なリソースが使用されるように、企業の IT 運用とデータ サイエンティストにとって非常に重要です。 NetAppテクノロジーに関する次の追加情報は、この検証の範囲外ですが、導入によっては関連する可能性があります。

ONTAPデータ管理ソフトウェアには、運用を合理化および簡素化し、総運用コストを削減するための次の機能が含まれています。

- インライン データ圧縮と拡張重複排除。データ圧縮によりストレージ ブロック内の無駄なスペースが削減され、重複排除により実効容量が大幅に増加します。これは、ローカルに保存されたデータとクラウドに階層化されたデータに適用されます。
- 最小、最大、および適応型サービス品質 (AQoS)。きめ細かなサービス品質 (QoS) 制御により、高度に共有された環境における重要なアプリケーションのパフォーマンス レベルを維持できます。
- NetAppFabricPool。 Amazon Web Services (AWS)、Azure、 NetApp StorageGRIDストレージ ソリューションなどのパブリックおよびプライベート クラウド ストレージ オプションへのコールド データの自動階層化を提供します。 FabricPoolの詳細については、以下を参照してください。 "TR-4598: FabricPool のベストプラクティス" 。

### データの高速化と保護

ONTAP は優れたレベルのパフォーマンスとデータ保護を提供し、これらの機能を次のように拡張します。

- パフォーマンスと低レイテンシ。 ONTAP は、可能な限り低いレイテンシで最高のスループットを提供します。
- データ保護：ONTAP は、すべてのプラットフォームにわたる共通管理を備えた組み込みのデータ保護機能を提供します。
- NetAppボリューム暗号化 (NVE)。 ONTAP は、オンボードと外部キー管理の両方をサポートするネイティブのボリューム レベルの暗号化を提供します。
- マルチテナントと多要素認証。 ONTAP は、最高レベルのセキュリティでインフラストラクチャ リソースを共有できるようにします。

### 将来を見据えたインフラ

ONTAP は、次の機能により、要求が厳しく常に変化するビジネス ニーズへの対応に役立ちます。

- シームレスなスケーリングと中断のない運用。 ONTAP は、既存のコントローラおよびスケールアウト クラスタへの無停止の容量追加をサポートします。顧客は、コストのかかるデータ移行や停止なしに、最新のテクノロジーにアップグレードできます。
- クラウド接続。 ONTAP は、すべてのパブリック クラウドのソフトウェア定義ストレージとクラウド ネイティブ インスタンスのオプションを備えた、最もクラウドに接続されたストレージ管理ソフトウェアで

す。

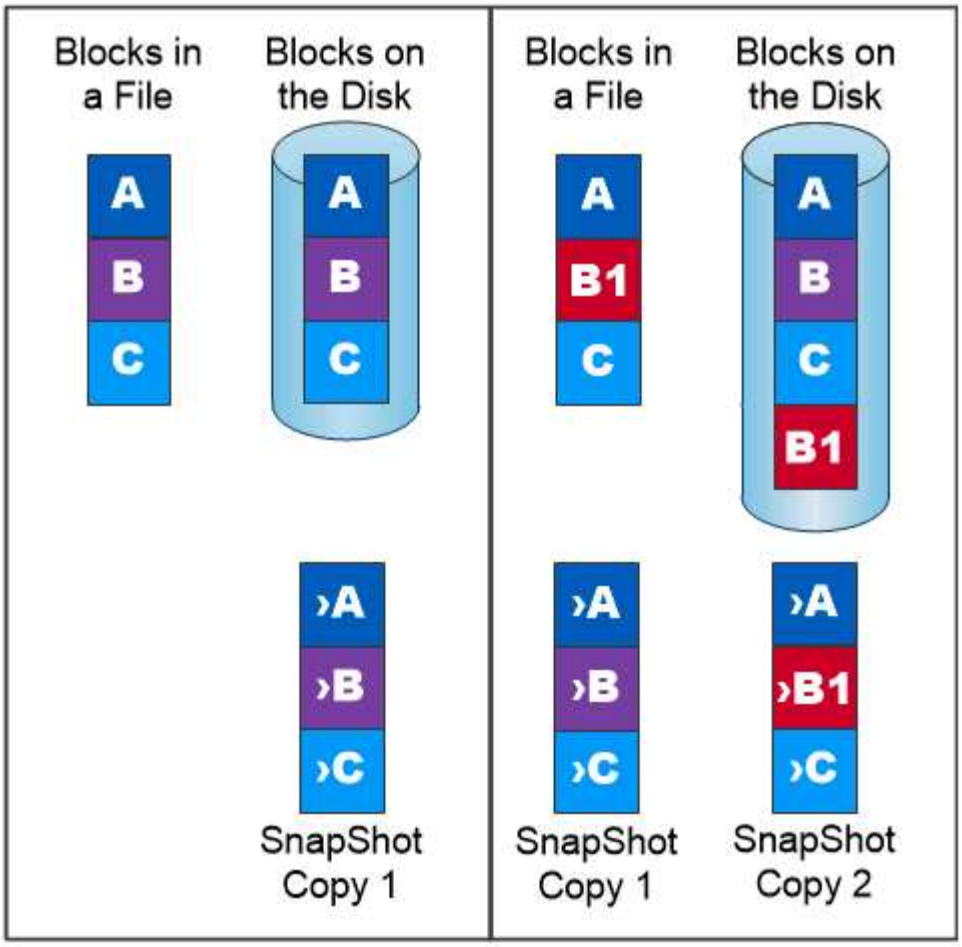
- 新しいアプリケーションとの統合。 ONTAP は、既存のエンタープライズ アプリケーションをサポートするのと同じインフラストラクチャを使用して、自律走行車、スマート シティ、インダストリー 4.0 などの次世代プラットフォームとアプリケーション向けにエンタープライズ グレードのデータ サービスを提供します。

## NetAppスナップショットコピー

NetAppスナップショット コピーは、ボリュームの読み取り専用の特定時点のイメージです。次の図に示すように、イメージは最後のスナップショット コピーの作成以降に作成されたファイルの変更のみを記録するため、消費するストレージ スペースは最小限で、パフォーマンスのオーバーヘッドもごくわずかです。

スナップショット コピーの効率性は、 ONTAP のコア ストレージ仮想化テクノロジである Write Anywhere File Layout (WAFL) によって実現されています。WAFLは、データベースのように、メタデータを使用してディスク上の実際のデータ ブロックを参照します。ただし、データベースとは異なり、既存のブロックは上書きされません。更新されたデータは新しいブロックに書き込まれ、メタデータが変更されます。 ONTAP は、データ ブロックをコピーするのではなく、Snapshot コピーを作成するときにメタデータを参照するため、Snapshot コピーは非常に効率的です。こうすることで、他のシステムがコピーするブロックを探すために要するシーク時間と、コピー自体の作成コストが削減されます。

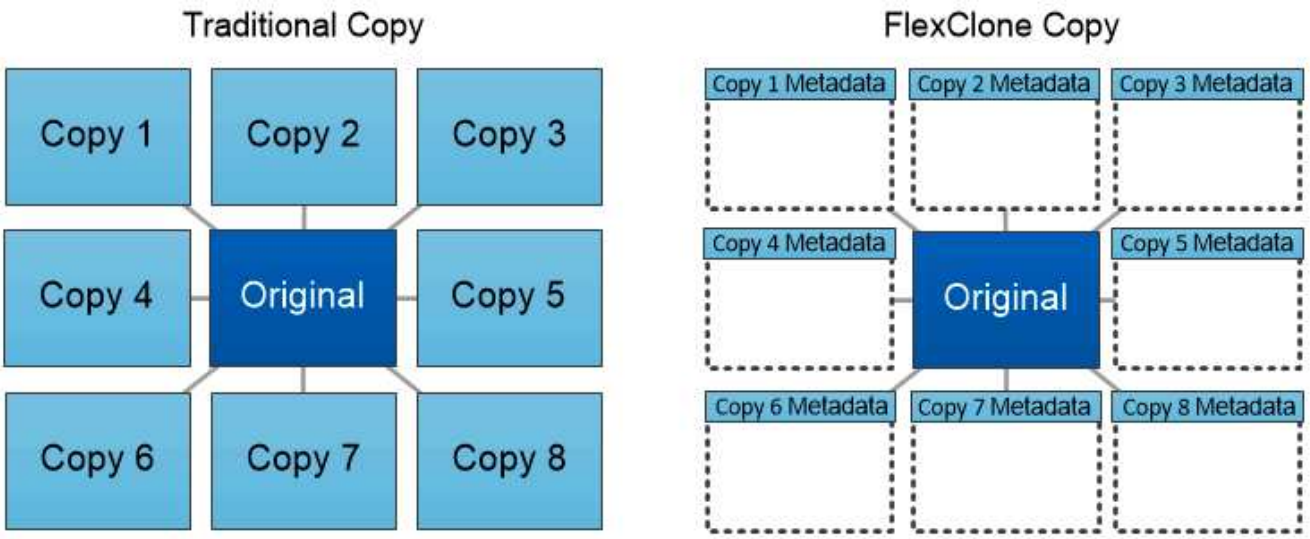
スナップショット コピーを使用すると、個々のファイルまたは LUN を回復したり、ボリュームの内容全体を復元したりできます。Snapshotコピーのポインタ情報をディスク上のデータと比較することで、ダウンタイムや多大なパフォーマンス コストなしで損失オブジェクトや破損オブジェクトが再構築されます。




*A Snapshot copy records only changes to the active file system since the last Snapshot copy.*

## NetApp FlexCloneテクノロジ

NetApp FlexCloneテクノロジーは、スナップショット メタデータを参照して、ボリュームの書き込み可能なポイントインタイム コピーを作成します。コピーは親とデータ ブロックを共有し、次の図に示すように、変更がコピーに書き込まれるまで、メタデータに必要なストレージ以外は消費しません。従来の手法でコピーを作成すると数分から数時間かかりますが、FlexCloneソフトウェアを使用すれば大規模なデータセットのコピーもほぼ瞬時に作成できます。そのため、同一のデータセットの複数のコピー (開発ワークスペースなど) やデータセットの一時的なコピー (本番データセットに対するアプリケーションのテストなど) が必要な状況に最適です。




*FlexClone copies share data blocks with their parents, consuming no storage except what is required for metadata.*

## NetApp SnapMirrorデータレプリケーションテクノロジー

NetApp SnapMirrorソフトウェアは、データ ファブリック全体にわたるコスト効率に優れ、使いやすい統合レプリケーション ソリューションです。 LAN または WAN 経由で高速にデータを複製します。仮想環境と従来の環境の両方で、ビジネスに不可欠なアプリケーションを含むあらゆる種類のアプリケーションに高いデータ可用性と高速なデータ複製を提供します。データを 1 つ以上のNetAppストレージ システムに複製し、セカンダリ データを継続的に更新すると、データは最新の状態に保たれ、必要なときにいつでも利用できるようになります。外部のレプリケーション サーバーは必要ありません。 SnapMirrorテクノロジーを活用したアーキテクチャの例については、次の図を参照してください。

SnapMirrorソフトウェアは、変更されたブロックのみをネットワーク経由で送信することで、 NetApp ONTAPストレージの効率を活用します。 SnapMirrorソフトウェアは、組み込みのネットワーク圧縮機能を使用して、データ転送を高速化し、ネットワーク帯域幅の使用率を最大 70% 削減します。 SnapMirrorテクノロジーを使用すると、1 つのシン レプリケーション データ ストリームを活用して、アクティブ ミラーと以前のポイントインタイム コピーの両方を保持する単一のリポジトリを作成し、ネットワーク トラフィックを最大 50% 削減できます。

## NetApp BlueXPコピーと同期

"BlueXPコピーと同期"高速かつ安全なデータ同期を実現するNetAppサービスです。オンプレミスの NFS または SMB ファイル共有、 NetApp StorageGRID、 NetApp ONTAP S3、 Google Cloud NetApp Volumes、 Azure NetApp Files、AWS S3、AWS EFS、Azure Blob、Google Cloud Storage、IBM Cloud Object Storage の間でファイルを転送する必要がある場合でも、 BlueXP Copy and Sync を使用すると、必要な場所にファイルを迅速かつ安全に移動します。

データが転送されると、ソースとターゲットの両方で完全に使用できるようになります。 BlueXP Copy and Sync は、更新がトリガーされたときにオンデマンドでデータを同期したり、事前定義されたスケジュールに基づいて継続的にデータを同期したりできます。いずれにしても、 BlueXP Copy and Sync はデルタのみを移動するため、データ複製にかかる時間とコストは最小限に抑えられます。

BlueXP Copy and Sync は、セットアップと使用が非常に簡単な SaaS (Software as a Service) ツールです。 BlueXP Copy and Sync によってトリガーされるデータ転送は、データ ブローカーによって実行されます。

BlueXPコピーおよび同期データブローカーは、AWS、Azure、Google Cloud Platform、またはオンプレミスにデプロイできます。

### NetApp XCP

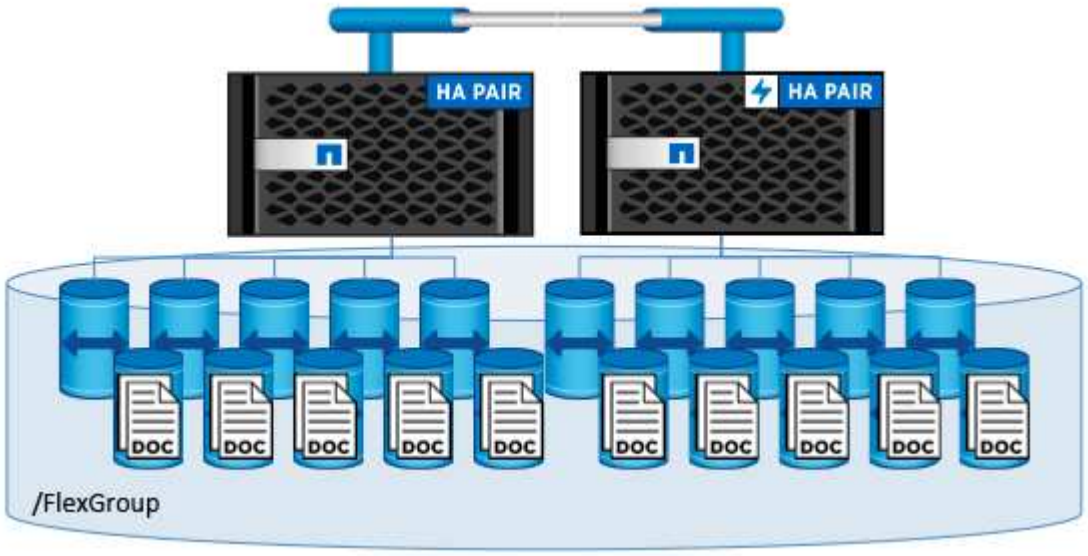
"NetApp XCP"は、any-to- NetAppおよびNetApp-to- NetAppのデータ移行とファイルシステム分析のためのクライアントベースのソフトウェアです。 XCP は、利用可能なすべてのシステム リソースを活用して大容量のデータセットと高パフォーマンスの移行を処理し、拡張して最大のパフォーマンスを実現するように設計されています。 XCP は、レポートを生成するオプションを使用して、ファイル システムの完全な可視性を得るのに役立ちます。

### NetApp ONTAP FlexGroupボリューム

トレーニング データセットは、数十億のファイルのコレクションになる可能性があります。ファイルには、並列に読み取るために保存および処理する必要があるテキスト、オーディオ、ビデオ、およびその他の形式の非構造化データが含まれる場合があります。ストレージ システムは、多数の小さなファイルを保存し、順次およびランダム I/O でそれらのファイルを並列に読み取る必要があります。

FlexGroupボリュームは、次の図に示すように、複数の構成メンバー ボリュームで構成される単一の名前空間です。ストレージ管理者の観点から見ると、 FlexGroupボリュームはNetApp FlexVol volumeのように管理され、動作します。 FlexGroupボリューム内のファイルは個々のメンバー ボリュームに割り当てられ、ボリュームまたはノード間でストライプ化されません。これらにより、次の機能が有効になります。

- FlexGroupボリュームは、高メタデータ ワークロードに対して、数ペタバイトの容量と予測可能な低レイテンシを提供します。
- 同じ名前空間で最大 4000 億のファイルをサポートします。
- これらは、CPU、ノード、アグリゲート、および構成するFlexVolボリューム全体にわたる NAS ワークロードでの並列操作をサポートします。




# アーキテクチャ

このソリューションは特定のハードウェアに依存しません。このソリューションは、

NetApp Tridentでサポートされているすべてのネットアップ物理ストレージ アプライアンス、ソフトウェア定義インスタンス、またはクラウド サービスと互換性があります。例としては、 NetApp AFFストレージ システム、 Amazon FSx ONTAP、 Azure NetApp Files、 Google Cloud NetApp Volumes、 NetApp Cloud Volumes ONTAPインスタンスなどが挙げられます。さらに、使用されている Kubernetes バージョンがNetApp Tridentおよび実装されているその他のソリューション コンポーネントでサポートされている限り、このソリューションは任意の Kubernetes クラスターに実装できます。 Tridentでサポートされている Kubernetes バージョンの一覧については、 "Tridentのドキュメント" 。このソリューションのさまざまなコンポーネントを検証するために使用された環境の詳細については、次の表を参照してください。

### Apache Airflow 検証環境

| ソフトウェアコンポーネント | **version** |
|---|---|
| Apacheエアフロー | 2.0.1、導入"Apache Airflow Helm チャート"8.0.8 |
| Kubernetes | 1.18 |
| NetAppTrident | 21.01 |

### JupyterHub 検証環境

| ソフトウェアコンポーネント | **version** |
|---|---|
| ジュピターハブ | 4.1.5、デプロイ経由"JupyterHub Helm チャート"3.3.7 |
| Kubernetes | 1.29 |
| NetAppTrident | 24.02 |

### MLflow 検証環境

| ソフトウェアコンポーネント | **version** |
|---|---|
| MLフロー | 2.14.1、以下でデプロイ"MLflow Helmチャート"1.4.12 |
| Kubernetes | 1.29 |
| NetAppTrident | 24.02 |

### Kubeflow 検証環境

| ソフトウェアコンポーネント | **version** |
|---|---|
| キューブフロー | 1.7、導入"デプロイKF"0.1.1 |
| Kubernetes | 1.26 |
| NetAppTrident | 23.07 |

サポート

NetApp は、Apache Airflow、JupyterHub、MLflow、Kubeflow、Kubernetes に対するエンタープライズ サポートを提供していません。完全にサポートされたMLOpsプラットフォームにご興味がございましたら、"ネットアップに連絡" NetApp がパートナーと共同で提供する、完全にサポートされた MLOps ソリューションについて説明します。

# NetApp Trident の構成

## NetApp AIPod導入におけるTridentバックエンドの例

Tridentを使用して Kubernetes クラスター内でストレージ リソースを動的にプロビジョニングする前に、1 つ以上のTridentバックエンドを作成する必要があります。以下の例は、このソリューションのコンポーネントをデプロイする場合に作成する可能性のあるさまざまなタイプのバックエンドを表しています。"NetApp AIPod" 。バックエンドの詳細、および他のプラットフォーム/環境のバックエンドの例については、"Tridentのドキュメント" 。

1. NetApp、 AIPod用にFlexGroup対応のTrident Backend を作成することを推奨しています。

   次のサンプルコマンドは、 AIPodストレージ仮想マシン (SVM) 用のFlexGroup対応Tridentバックエンドの作成を示しています。このバックエンドは `ontap-nas-flexgroup`ストレージ ドライバー。 ONTAP は、FlexVolとFlexGroup2 つの主要なデータ ボリューム タイプをサポートしています。 FlexVolボリュームにはサイズ制限があります (この執筆時点では、最大サイズは特定の展開によって異なります)。一方、FlexGroupボリュームは、最大 20PB および 4000 億ファイルまで直線的に拡張でき、単一の名前空間を提供することでデータ管理を大幅に簡素化します。したがって、 FlexGroupボリュームは、大量のデータに依存する AI および ML ワークロードに最適です。

   少量のデータで作業していて、 FlexGroupボリュームの代わりにFlexVolボリュームを使用する場合は、`ontap-nas`ストレージドライバーの代わりに `ontap-nas-flexgroup`ストレージ ドライバー。

```
$ cat << EOF > ./trident-backend-aipod-flexgroups-iface1.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "aipod-flexgroups-iface1",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-aipod-flexgroups-
iface1.json -n trident
+------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME         |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+------------------------+---------------------
+--------------------------------------+--------+---------+
| aipod-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-
b263-b6da6dec0bdd | online |       0 |
+------------------------+---------------------
+--------------------------------------+--------+---------+
$ tridentctl get backend -n trident
+------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME         |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+------------------------+---------------------
+--------------------------------------+--------+---------+
| aipod-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-
b263-b6da6dec0bdd | online |       0 |
+------------------------+---------------------
+--------------------------------------+--------+---------+
```

2. NetApp、 FlexVol対応のTridentバックエンドの作成も推奨しています。永続的なアプリケーションのホスティング、結果、出力、デバッグ情報などの保存には、 FlexVolボリュームを使用することをお勧めします。 FlexVolボリュームを使用する場合は、 FlexVol対応のTridentバックエンドを 1 つ以上作成する必要があります。次のサンプル コマンドは、単一のFlexVol対応Tridentバックエンドの作成を示しています。

```
$ cat << EOF > ./trident-backend-aipod-flexvols.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "aipod-flexvols",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-aipod-flexvols.json -n
trident
+------------------------+--------------------
+--------------------------------------+--------+---------+
|            NAME         |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+------------------------+--------------------
+--------------------------------------+--------+---------+
| aipod-flexvols          | ontap-nas           | 52bdb3b1-13a5-4513-a9c1-
52a69657fabe | online |       0 |
+------------------------+--------------------
+--------------------------------------+--------+---------+
$ tridentctl get backend -n trident
+------------------------+--------------------
+--------------------------------------+--------+---------+
|            NAME         |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+------------------------+--------------------
+--------------------------------------+--------+---------+
| aipod-flexvols          | ontap-nas           | 52bdb3b1-13a5-4513-a9c1-
52a69657fabe | online |       0 |
| aipod-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-b263-
b6da6dec0bdd | online |       0 |
+------------------------+--------------------
+--------------------------------------+--------+---------+
```

### NetApp AIPodデプロイメント用の Kubernetes ストレージクラスの例

Tridentを使用して Kubernetes クラスター内でストレージ リソースを動的にプロビジョニングする前に、1 つ以上の Kubernetes StorageClasses を作成する必要があります。以下の例は、このソリューションのコンポーネントを次の場所に展開する場合に作成する可能性のあるさまざまなタイプのストレージクラスを表しています。"NetApp AIPod"

。 StorageClassesの詳細、および他のプラットフォーム/環境のStorageClassesの例については、"Tridentのドキュメント" 。

1. NetAppは、セクションで作成したFlexGroup対応Tridentバックエンド用のStorageClassを作成することを推奨します。"NetApp AIPod導入におけるTridentバックエンドの例" 、ステップ 1。以下のコマンド例は、セクションで作成されたサンプルバックエンドに対応する複数のストレージクラスの作成を示しています。"NetApp AIPod導入におけるTridentバックエンドの例"ステップ1 - 活用するもの"NFS over RDMA"そしてそうではないもの。

   対応するPersistentVolumeClaim（PVC）が削除されたときに永続ボリュームが削除されないように、次の例では `reclaimPolicy`の価値 `Retain`。詳細については、 `reclaimPolicy`フィールドについては、公式 "Kubernetesドキュメント"。

   注: 次の例の StorageClasses では、最大転送サイズとして 262144 を使用します。この最大転送サイズを使用するには、 ONTAPシステムで最大転送サイズを適切に設定する必要があります。参照"ONTAPのドキュメント"詳細については。

   注: NFS over RDMA を使用するには、 ONTAPシステムで NFS over RDMA を設定する必要があります。参照"ONTAPのドキュメント"詳細については。

   注: 次の例では、StorageClass 定義ファイルの storagePool フィールドに特定のバックエンドが指定されています。

```
$ cat << EOF > ./storage-class-aipod-flexgroups-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aipod-flexgroups-retain
provisioner: csi.trident.netapp.io
mountOptions: ["vers=4.1", "nconnect=16", "rsize=262144",
"wsize=262144"]
parameters:
  backendType: "ontap-nas-flexgroup"
  storagePools: "aipod-flexgroups-iface1:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-aipod-flexgroups-retain.yaml
storageclass.storage.k8s.io/aipod-flexgroups-retain created
$ cat << EOF > ./storage-class-aipod-flexgroups-retain-rdma.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aipod-flexgroups-retain-rdma
provisioner: csi.trident.netapp.io
mountOptions: ["vers=4.1", "proto=rdma", "max_connect=16",
"rsize=262144", "wsize=262144"]
parameters:
  backendType: "ontap-nas-flexgroup"
  storagePools: "aipod-flexgroups-iface1:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-aipod-flexgroups-retain-rdma.yaml
storageclass.storage.k8s.io/aipod-flexgroups-retain-rdma created
$ kubectl get storageclass
NAME                              PROVISIONER             AGE
aipod-flexgroups-retain           csi.trident.netapp.io   0m
aipod-flexgroups-retain-rdma      csi.trident.netapp.io   0m
```

2. NetApp、セクションで作成したFlexVol対応Tridentバックエンドに対応するStorageClassを作成することも推奨しています。"AIPodデプロイメント用のTridentバックエンドの例" 、ステップ2。次のサンプルコマンドは、 FlexVolボリューム用の単一の StorageClass の作成を示しています。

   注: 次の例では、StorageClass 定義ファイルの storagePool フィールドに特定の Backend が指定されていません。このStorageClassを使用してKubernetesでボリュームを管理する場合、 Tridentは利用可能なバックエンドを使用しようとします。 `ontap-nas`ドライバ。

```
$ cat << EOF > ./storage-class-aipod-flexvols-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aipod-flexvols-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-aipod-flexvols-retain.yaml
storageclass.storage.k8s.io/aipod-flexvols-retain created
$ kubectl get storageclass
NAME                              PROVISIONER             AGE
aipod-flexgroups-retain           csi.trident.netapp.io   0m
aipod-flexgroups-retain-rdma      csi.trident.netapp.io   0m
aipod-flexvols-retain             csi.trident.netapp.io   0m
```

# Apacheエアフロー

## Apache Airflow デプロイメント

このセクションでは、Kubernetes クラスターに Airflow をデプロイするために完了する必要があるタスクについて説明します。

Airflow は Kubernetes 以外のプラットフォームにもデプロイできます。 Kubernetes 以外のプラットフォームに Airflow をデプロイすることは、このソリューションの範囲外です。

前提条件

このセクションで説明する展開演習を実行する前に、次のタスクが既に実行されていることを前提としています。

1. すでに動作中の Kubernetes クラスターがあります。
2. Kubernetes クラスターにNetApp Tridentがすでにインストールされ、構成されています。Tridentの詳細については、"Tridentのドキュメント" 。

### Helmをインストールする

Airflow は、Kubernetes の一般的なパッケージ マネージャーである Helm を使用してデプロイされます。Airflow をデプロイする前に、デプロイ ジャンプ ホストに Helm をインストールする必要があります。デプロイメントジャンプホストにHelmをインストールするには、 "インストール手順"公式 Helm ドキュメントに記載されています。

### デフォルトの**Kubernetes**ストレージクラスを設定する

Airflow をデプロイする前に、Kubernetes クラスター内でデフォルトの StorageClass を指定する必要があります。 Airflow のデプロイメント プロセスでは、デフォルトの StorageClass を使用して新しい永続ボリュームをプロビジョニングしようとします。デフォルトの StorageClass として StorageClass が指定されていない場合、デプロイメントは失敗します。クラスター内でデフォルトのストレージクラスを指定するには、"Kubeflow デプロイメント"セクション。クラスター内でデフォルトの StorageClass をすでに指定している場合は、この手順をスキップできます。

### **Helm** を使用して **Airflow** をデプロイする

Helm を使用して Kubernetes クラスターに Airflow をデプロイするには、デプロイメント ジャンプ ホストから次のタスクを実行します。

1. Helmを使用してAirflowをデプロイするには、 "展開手順" Artifact Hub の公式 Airflow チャートをご覧ください。次のサンプルコマンドは、Helm を使用した Airflow のデプロイメントを示しています。値を変更、追加、削除します。 `custom- values.yaml`環境と希望する構成に応じて、必要に応じてファイルを変更します。

```
$ cat << EOF > custom-values.yaml
#####################################
# Airflow - Common Configs
#####################################
airflow:
  ## the airflow executor type to use
  ##
  executor: "CeleryExecutor"
  ## environment variables for the web/scheduler/worker Pods (for
airflow configs)
  ##
  #
#####################################
# Airflow - WebUI Configs
#####################################
web:
  ## configs for the Service of the web Pods
  ##
  service:
    type: NodePort
#####################################
# Airflow - Logs Configs
#####################################
logs:
  persistence:
    enabled: true
#####################################
# Airflow - DAGs Configs
#####################################
```

```
dags:
  ## configs for the DAG git repository & sync container
  ##
  gitSync:
    enabled: true
    ## url of the git repository
    ##
    repo: "git@github.com:mboglesby/airflow-dev.git"
    ## the branch/tag/sha1 which we clone
    ##
    branch: master
    revision: HEAD
    ## the name of a pre-created secret containing files for ~/.ssh/
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for SSH git repos
    ## - the secret commonly includes files: id_rsa, id_rsa.pub,
known_hosts
    ## - known_hosts is NOT NEEDED if `git.sshKeyscan` is true
    ##
    sshSecret: "airflow-ssh-git-secret"
    ## the name of the private key file in your `git.secret`
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for PRIVATE SSH git repos
    ##
    sshSecretKey: id_rsa
    ## the git sync interval in seconds
    ##
    syncWait: 60
EOF
$ helm install airflow airflow-stable/airflow -n airflow --version 8.0.8
--values ./custom-values.yaml
...
Congratulations. You have just deployed Apache Airflow!
1. Get the Airflow Service URL by running these commands:
   export NODE_PORT=$(kubectl get --namespace airflow -o
jsonpath="{.spec.ports[0].nodePort}" services airflow-web)
   export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
   echo http://$NODE_IP:$NODE_PORT/
2. Open Airflow in your web browser
```

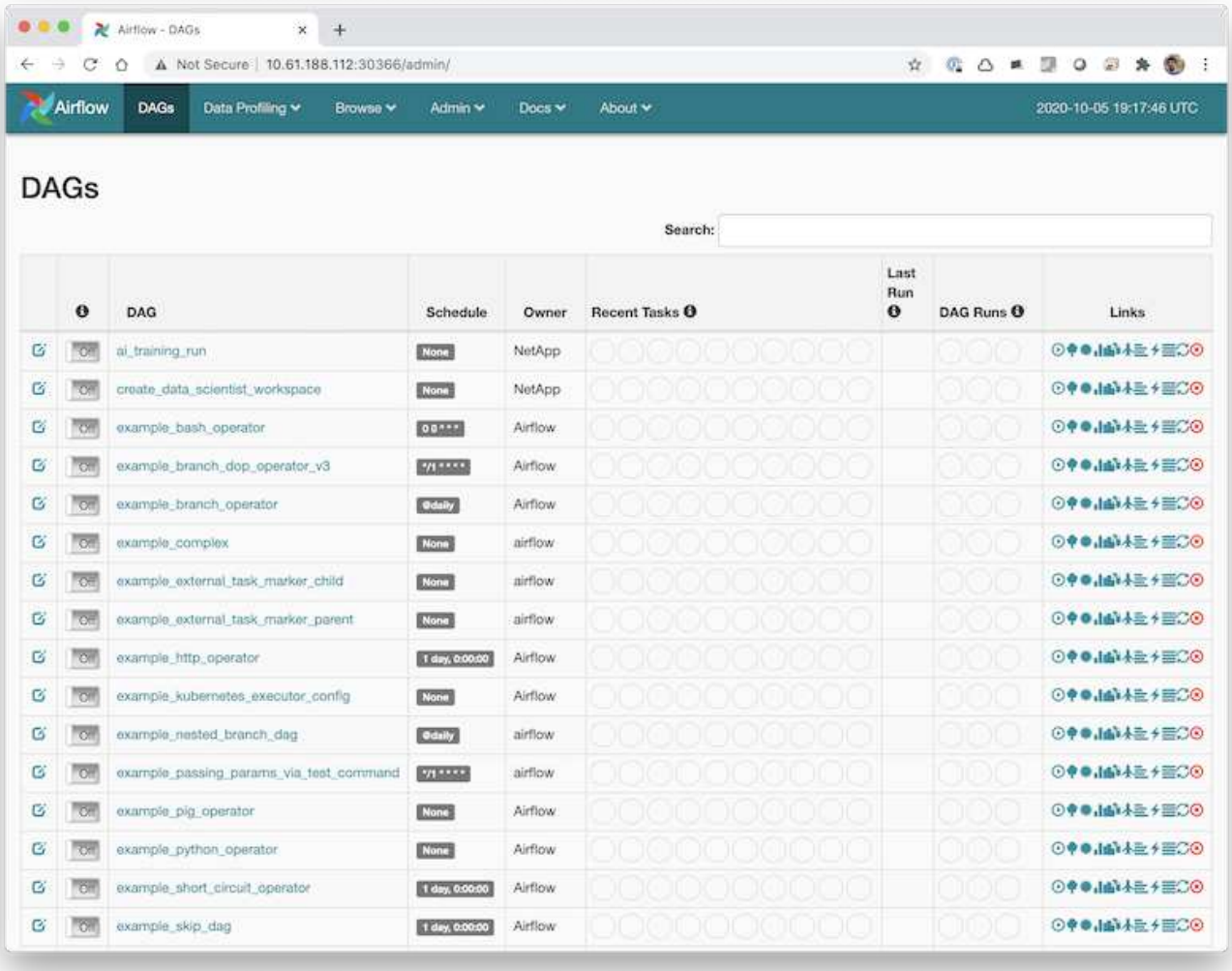
2. すべての Airflow ポッドが稼働していることを確認します。すべてのポッドが起動するまでに数分かかる場合があります。

```
$ kubectl -n airflow get pod
NAME                                  READY   STATUS    RESTARTS   AGE
airflow-flower-b5656d44f-h8qjk        1/1     Running   0          2h
airflow-postgresql-0                  1/1     Running   0          2h
airflow-redis-master-0                1/1     Running   0          2h
airflow-scheduler-9d95fcdf9-clf4b     2/2     Running   2          2h
airflow-web-59c94db9c5-z7rg4          1/1     Running   0          2h
airflow-worker-0                      2/2     Running   2          2h
```

3. 手順 1 で Helm を使用して Airflow をデプロイしたときにコンソールに表示された手順に従って、Airflow Web サービス URL を取得します。

```
$ export NODE_PORT=$(kubectl get --namespace airflow -o
jsonpath="{.spec.ports[0].nodePort}" services airflow-web)
$ export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
$ echo http://$NODE_IP:$NODE_PORT/
```

4. Airflow Web サービスにアクセスできることを確認します。




### NetApp DataOps Toolkit を Airflow と併用する

その "Kubernetes 用NetApp DataOps ツールキット"Airflowと併用できます。 NetApp DataOps Toolkit を Airflow と併用すると、スナップショットやクローンの作成などのNetAppデータ管理操作を、Airflow によってオーケストレーションされる自動化されたワークフローに組み込むことができます。

参照 "空気の流れの例"Airflow でツールキットを使用する方法の詳細については、 NetApp DataOps Toolkit GitHub リポジトリ内のセクションを参照してください。

## ジュピターハブ

### JupyterHub デプロイメント

このセクションでは、Kubernetes クラスターに JupyterHub をデプロイするために完了する必要があるタスクについて説明します。

JupyterHub は Kubernetes 以外のプラットフォームにもデプロイできます。 Kubernetes 以外のプラットフォームに JupyterHub をデプロイすることは、このソリューションの範囲外です。

### 前提条件

このセクションで説明する展開演習を実行する前に、次のタスクが既に実行されていることを前提としています。

1. すでに動作中の Kubernetes クラスターがあります。
2. Kubernetes クラスターにNetApp Tridentがすでにインストールされ、構成されています。Tridentの詳細については、"Tridentのドキュメント" 。

### **Helm**をインストールする

JupyterHub は、Kubernetes の一般的なパッケージ マネージャーである Helm を使用してデプロイされます。 JupyterHub をデプロイする前に、Kubernetes コントロール ノードに Helm をインストールする必要があります。 Helmをインストールするには、 "インストール手順"公式 Helm ドキュメントに記載されています。

### デフォルトの**Kubernetes**ストレージクラスを設定する

JupyterHub をデプロイする前に、Kubernetes クラスター内でデフォルトの StorageClass を指定する必要があります。クラスター内でデフォルトのストレージクラスを指定するには、"Kubeflow デプロイメント"セクション。クラスター内でデフォルトの StorageClass をすでに指定している場合は、この手順をスキップできます。

### **JupyterHub**をデプロイする

上記の手順を完了すると、JupyterHub をデプロイする準備が整います。 JupyterHub のデプロイメントには次の手順が必要です。

#### **JupyterHub** デプロイメントを構成する

デプロイメントの前に、それぞれの環境に合わせて JupyterHub デプロイメントを最適化することをお勧めします。 **config.yaml** ファイルを作成し、Helm チャートを使用してデプロイ中にそれを利用できます。

**config.yaml** ファイルの例は次の場所にあります。 https://github.com/jupyterhub/zero-to-jupyterhub-k8s/blob/HEAD/jupyterhub/values.yaml

この config.yaml ファイルでは、 NetApp Trident StorageClass の **(singleuser.storage.dynamic.storageClass)** パラメータを設定できます。これは、個々のユーザー ワークスペースのボリュームをプロビジョニングするために使用されるストレージ クラスです。

#### 共有ボリュームの追加

すべての JupyterHub ユーザーに共有ボリュームを使用する場合は、それに応じて **config.yaml** を調整できます。たとえば、jupyterhub-shared-volume という共有 PersistentVolumeClaim がある場合、次のようにすべてのユーザー ポッドで /home/shared としてマウントできます。

```
singleuser:
  storage:
    extraVolumes:
      - name: jupyterhub-shared
        persistentVolumeClaim:
          claimName: jupyterhub-shared-volume
    extraVolumeMounts:
      - name: jupyterhub-shared
        mountPath: /home/shared
```

これはオプションの手順であり、これらのパラメータは必要に応じて調整できます。

#### Helm ChartでJupyterHubをデプロイする

Helm に JupyterHub Helm チャート リポジトリを認識させます。

```
helm repo add jupyterhub https://hub.jupyter.org/helm-chart/
helm repo update
```

次のような出力が表示されます。

```
Hang tight while we grab the latest from your chart repositories...
...Skip local chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "jupyterhub" chart repository
Update Complete. □ Happy Helming!□
```

次に、config.yaml が含まれているディレクトリから次のコマンドを実行して、config.yaml で構成されたチャートをインストールします。

```
helm upgrade --cleanup-on-fail \
  --install my-jupyterhub jupyterhub/jupyterhub \
  --namespace my-namespace \
  --create-namespace \
  --values config.yaml
```

この例では、

<helm-release-name> は my-jupyterhub に設定されており、これが JupyterHub リリースの名前になります。 <k8s-namespace> は、JupyterHub をインストールする名前空間である my-namespace に設定されています。 --create-namespace フラグは、名前空間がまだ存在しない場合に名前空間を作成するために使用されます。 --values フラグは、必要な構成オプションを含む config.yaml ファイルを指定します。

デプロイメントの確認

ステップ 2 の実行中に、次のコマンドでポッドが作成されていることを確認できます。

```
kubectl get pod --namespace <k8s-namespace>
```

ハブとプロキシ ポッドが実行状態になるまで待ちます。

```
NAME                      READY     STATUS    RESTARTS   AGE
hub-5d4ffd57cf-k68z8      1/1       Running   0          37s
proxy-7cb9bc4cc-9bdlp     1/1       Running   0          37s
```

**JupyterHub** にアクセスする

JupyterHub にアクセスするために使用できる IP を見つけます。出力例のように、proxy-public サービスの EXTERNAL-IP が使用可能になるまで、次のコマンドを実行します。

config.yaml ファイルで NodePort サービスを使用しましたが、設定に基づいて環境に合わせて調整できます (例: LoadBalancer)。

```
kubectl --namespace <k8s-namespace> get service proxy-public
```

```
NAME           TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
AGE
proxy-public   NodePort   10.51.248.230   104.196.41.97   80:30000/TCP
1m
```

JupyterHub を使用するには、ブラウザにプロキシ パブリック サービスの外部 IP を入力します。

## JupyterHubでNetApp DataOpsツールキットを使用する

その "Kubernetes 用NetApp DataOps ツールキット"JupyterHub と組み合わせて使用できます。 NetApp DataOps Toolkit を JupyterHub と併用すると、エンド ユーザーは、ワークスペースのバックアップやデータセットからモデルへのトレーサビリティのためのボリューム スナップショットを Jupyter Notebook 内から直接作成できるようになります。

初期セットアップ

DataOps Toolkit を JupyterHub で使用する前に、JupyterHub が個々のユーザーの Jupyter Notebook Server ポッドに割り当てる Kubernetes サービス アカウントに適切な権限を付与する必要があります。 JupyterHub は、 `singleuser.serviceAccountName` JupyterHub Helm チャート構成ファイルの変数。

**DataOps** ツールキットのクラスターロールを作成する

まず、ボリューム スナップショットを作成するために必要な Kubernetes API 権限を持つ「netapp-dataops」という名前のクラスター ロールを作成します。

```
$ vi clusterrole-netapp-dataops-snapshots.yaml
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: netapp-dataops-snapshots
rules:
- apiGroups: [""]
  resources: ["persistentvolumeclaims", "persistentvolumeclaims/status",
"services"]
  verbs: ["get", "list"]
- apiGroups: ["snapshot.storage.k8s.io"]
  resources: ["volumesnapshots", "volumesnapshots/status",
"volumesnapshotcontents", "volumesnapshotcontents/status"]
  verbs: ["get", "list", "create"]

$ kubectl create -f clusterrole-netapp-dataops-snapshots.yaml
clusterrole.rbac.authorization.k8s.io/netapp-dataops-snapshots created
```

ノートブック サーバーのサービス アカウントにクラスター ロールを割り当てる

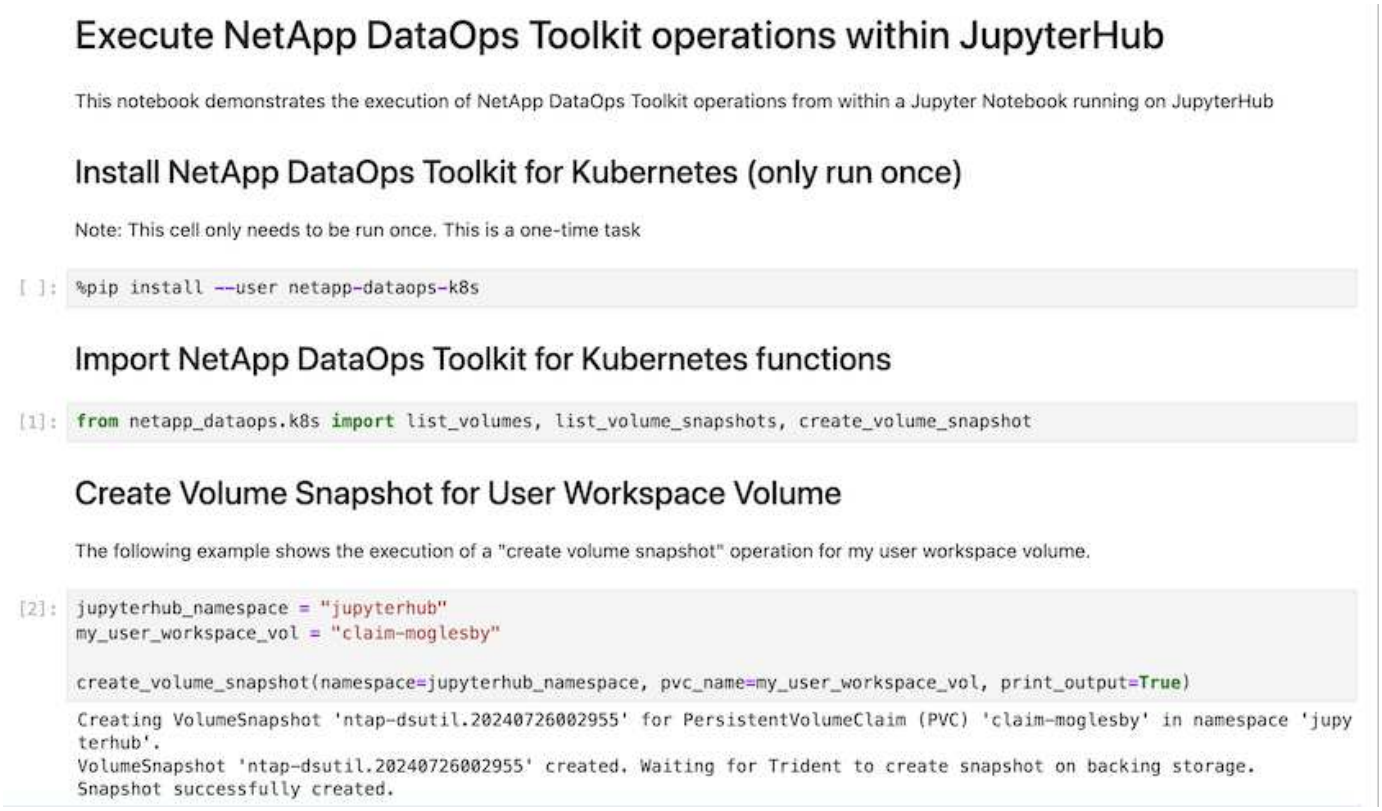
適切な名前空間内の適切なサービス アカウントに 'netapp-dataops-snapshots' クラスター ロールを割り当てるロール バインディングを作成します。たとえば、JupyterHubを「jupyterhub」名前空間にインストールし、`singleuser.serviceAccountName`変数を使用する場合は、次の例に示すように、 'netapp-dataops-snapshots' クラスター ロールを 'jupyterhub' 名前空間の 'default' サービス アカウントに割り当てます。

```
$ vi rolebinding-jupyterhub-netapp-dataops-snapshots.yaml
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jupyterhub-netapp-dataops-snapshots
  namespace: jupyterhub # Replace with you JupyterHub namespace
subjects:
- kind: ServiceAccount
  name: default # Replace with your JupyterHub
singleuser.serviceAccountName
  namespace: jupyterhub # Replace with you JupyterHub namespace
roleRef:
  kind: ClusterRole
  name: netapp-dataops-snapshots
  apiGroup: rbac.authorization.k8s.io

$ kubectl create -f ./rolebinding-jupyterhub-netapp-dataops-snapshots.yaml
rolebinding.rbac.authorization.k8s.io/jupyterhub-netapp-dataops-snapshots
created
```

**Jupyter Notebook** 内でボリューム スナップショットを作成する

現在、JupyterHub ユーザーは、次の例に示すように、 NetApp DataOps Toolkit を使用して、Jupyter Notebook 内から直接ボリューム スナップショットを作成できます。




### NetApp SnapMirrorを使用してJupyterHubにデータを取り込む

NetApp SnapMirrorは、 NetAppストレージ システム間でデータを複製できるレプリケーション テクノロジです。 SnapMirror を使用すると、リモート環境から JupyterHub にデータを取り込むことができます。

ワークフローとデモの例

参照"このTech ONTAPブログ投稿"NetApp SnapMirrorを使用して JupyterHub にデータを取り込む詳細なワークフロー例とデモをご覧ください。

# MLフロー

### MLflow デプロイメント

このセクションでは、Kubernetes クラスターに MLflow をデプロイするために完了する必要があるタスクについて説明します。

MLflow は Kubernetes 以外のプラットフォームにもデプロイできます。 Kubernetes 以外のプラットフォームに MLflow をデプロイすることは、このソリューションの範囲外です。

前提条件

このセクションで説明する展開演習を実行する前に、次のタスクが既に実行されていることを前提としています。

1. すでに動作中の Kubernetes クラスターがあります。
2. Kubernetes クラスターにNetApp Tridentがすでにインストールされ、構成されています。Tridentの詳細については、"Tridentのドキュメント" 。

### Helmをインストールする

MLflow は、Kubernetes の一般的なパッケージ マネージャーである Helm を使用してデプロイされます。MLflow をデプロイする前に、Kubernetes コントロール ノードに Helm をインストールする必要があります。Helmをインストールするには、 "インストール手順"公式 Helm ドキュメントに記載されています。

### デフォルトのKubernetesストレージクラスを設定する

MLflow をデプロイする前に、Kubernetes クラスター内でデフォルトの StorageClass を指定する必要があります。クラスター内でデフォルトのストレージクラスを指定するには、"Kubeflow デプロイメント"セクション。クラスター内でデフォルトの StorageClass をすでに指定している場合は、この手順をスキップできます。

### MLflowをデプロイする

前提条件が満たされたら、Helm チャートを使用して MLflow のデプロイを開始できます。

#### MLflow Helm Chart のデプロイメントを構成します。

Helm チャートを使用して MLflow をデプロイする前に、**config.yaml** ファイルを使用して、 NetApp Trident ストレージ クラスを使用するようにデプロイを構成し、ニーズに合わせて他のパラメータを変更できます。**config.yaml** ファイルの例は、次の場所にあります。 https://github.com/bitnami/charts/blob/main/bitnami/mlflow/values.yaml

> config.yaml ファイルの **global.defaultStorageClass** パラメータでTrident storageClass を設定できます (例: storageClass: "ontap-flexvol")。

#### Helm Chartのインストール

Helm チャートは、次のコマンドを使用して、MLflow のカスタム **config.yaml** ファイルとともにインストールできます。

```
helm install oci://registry-1.docker.io/bitnamicharts/mlflow -f
config.yaml --generate-name --namespace jupyterhub
```

> このコマンドは、提供された **config.yaml** ファイルを介してカスタム構成で Kubernetes クラスターに MLflow をデプロイします。 MLflow は指定された名前空間にデプロイされ、リリースに対して Kubernetes 経由でランダムなリリース名が与えられます。

#### デプロイメントの確認

Helm チャートのデプロイが完了したら、次のコマンドを使用してサービスにアクセスできるかどうかを確認できます。

```
kubectl get service -n jupyterhub
```

**jupyterhub** を、デプロイ時に使用した名前空間に置き換えます。

次のサービスが表示されます。

```
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)             AGE
mlflow-1719843029-minio             ClusterIP   10.233.22.4     <none>
80/TCP,9001/TCP     25d
mlflow-1719843029-postgresql        ClusterIP   10.233.5.141    <none>
5432/TCP            25d
mlflow-1719843029-postgresql-hl     ClusterIP   None            <none>
5432/TCP            25d
mlflow-1719843029-tracking          NodePort    10.233.2.158    <none>
30002:30002/TCP     25d
```

NodePort サービスを使用してポート 30002 の MLflow にアクセスするように config.yaml ファイルを編集しました。

**MLflow** にアクセスする

MLflowに関連するすべてのサービスが起動したら、指定されたNodePortまたはLoadBalancerのIPアドレス（例： `http://10.61.181.109:30002` ）

## NetAppと MLflow によるデータセットからモデルへのトレーサビリティ

その "Kubernetes 用NetApp DataOps ツールキット"データセットからモデル、またはワークスペースからモデルへのトレーサビリティを実装するために、MLflow の実験追跡機能と組み合わせて使用できます。

データセットからモデルへ、またはワークスペースからモデルへのトレーサビリティを実装するには、次のサンプルコード スニペットに示すように、トレーニング実行の一部として DataOps ツールキットを使用してデータセットまたはワークスペース ボリュームのスナップショットを作成します。このコードは、MLflow 実験追跡サーバーに記録する特定のトレーニング実行に関連付けられたタグとして、データ ボリューム名とスナップショット名を保存します。

```
...
from netapp_dataops.k8s import create_volume_snapshot

with mlflow.start_run() :
    ...

    namespace = "my_namespace" # Kubernetes namespace in which dataset
volume PVC resides
    dataset_volume_name = "project1" # Name of PVC corresponding to
dataset volume
    snapshot_name = "run1" # Name to assign to your new snapshot

    # Create snapshot
    create_volume_snapshot(
        namespace=namespace,
        pvc_name=dataset_volume_name,
        snapshot_name=snapshot_name,
        printOutput=True
    )

    # Log data volume name and snapshot name as "tags"
    # associated with this training run in mlflow.
    mlflow.set_tag("data_volume_name", dataset_volume_name)
    mlflow.set_tag("snapshot_name", snapshot_name)

    ...
```

# キューブフロー

## Kubeflow デプロイメント

このセクションでは、Kubernetes クラスターに Kubeflow をデプロイするために完了する必要があるタスクについて説明します。

前提条件

このセクションで説明する展開演習を実行する前に、次のタスクが既に実行されていることを前提としています。

1. すでに稼働中の Kubernetes クラスターがあり、デプロイする予定の Kubeflow バージョンでサポートされている Kubernetes のバージョンを実行しています。サポートされているKubernetesのバージョンのリストについては、Kubeflowバージョンの依存関係を参照してください。"Kubeflowの公式ドキュメント" 。
2. Kubernetes クラスターにNetApp Tridentがすでにインストールされ、構成されています。Tridentの詳細については、"Tridentのドキュメント" 。

### デフォルトの**Kubernetes**ストレージクラスを設定する

Kubeflow をデプロイする前に、Kubernetes クラスター内でデフォルトの StorageClass を指定することをお勧めします。 Kubeflow のデプロイメント プロセスでは、デフォルトの StorageClass を使用して新しい永続ボリュームのプロビジョニングが試行される場合があります。デフォルトの StorageClass として StorageClass が指定されていない場合、デプロイメントは失敗する可能性があります。クラスター内でデフォルトの StorageClass を指定するには、デプロイメント ジャンプ ホストから次のタスクを実行します。クラスター内でデフォルトの StorageClass をすでに指定している場合は、この手順をスキップできます。

1. 既存の StorageClass の 1 つをデフォルトの StorageClass として指定します。以下のコマンド例は、StorageClassの指定を示しています。 `ontap-ai-flexvols-retain`デフォルトの StorageClass として。

> その `ontap-nas-flexgroup`Tridentバックエンド タイプは、最小 `PVC` サイズがかなり大きくなります。デフォルトでは、`Kubeflow` はサイズが数 `GB` しかない `PVC` をプロビジョニングしようとします。したがって、 `ontap-nas-flexgroup` Kubeflow デプロイメントの目的で、デフォルトの StorageClass としてバックエンド タイプを指定します。

```
$ kubectl get sc
NAME                                  PROVISIONER             AGE
ontap-ai-flexgroups-retain            csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface1     csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface2     csi.trident.netapp.io   25h
ontap-ai-flexvols-retain              csi.trident.netapp.io   3s
$ kubectl patch storageclass ontap-ai-flexvols-retain -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/ontap-ai-flexvols-retain patched
$ kubectl get sc
NAME                                   PROVISIONER             AGE
ontap-ai-flexgroups-retain             csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface1      csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface2      csi.trident.netapp.io   25h
ontap-ai-flexvols-retain (default)     csi.trident.netapp.io   54s
```

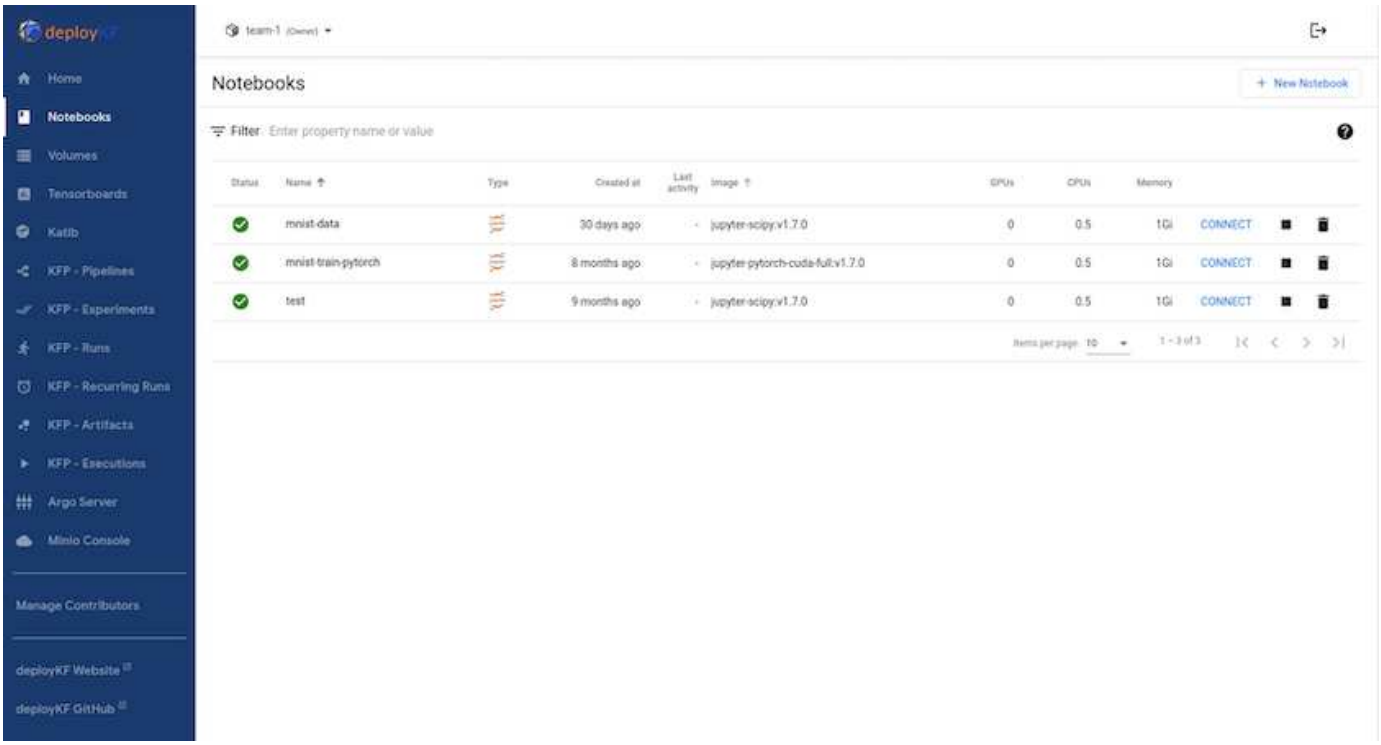
### **Kubeflow** のデプロイメント オプション

Kubeflow をデプロイするにはさまざまなオプションがあります。参照"Kubeflowの公式ドキュメント"展開オプションのリストを確認し、ニーズに最適なオプションを選択してください。

> 検証のために、Kubeflow 1.7をデプロイしました。"デプロイKF" 0.1.1。

## データ サイエンティストまたは開発者向けに **Jupyter Notebook** ワークスペースをプロビジョニングする

Kubeflow は、データ サイエンティストのワークスペースとして機能する新しい Jupyter Notebook サーバーを迅速にプロビジョニングできます。 Kubeflowコンテキスト内のJupyter Notebookの詳細については、 "Kubeflowの公式ドキュメント" 。




## KubeflowでNetApp DataOpsツールキットを使用する

その "Kubernetes 向けNetAppデータ サイエンス ツールキット"Kubeflow と組み合わせて使用できます。 NetApp Data Science Toolkit を Kubeflow と併用すると、次のような利点があります。

- データ サイエンティストは、スナップショットやクローンの作成など、高度なNetAppデータ管理操作を Jupyter Notebook 内から直接実行できます。
- スナップショットやクローンの作成などの高度なNetAppデータ管理操作は、Kubeflow Pipelines フレームワークを使用して自動化されたワークフローに組み込むことができます。

参照 "Kubeflowの例"Kubeflow でツールキットを使用する方法の詳細については、 NetApp Data Science Toolkit GitHub リポジトリ内のセクションを参照してください。

## ワークフロー例 - Kubeflow とNetApp DataOps Toolkit を使用して画像認識モデルをトレーニングする

このセクションでは、Kubeflow とNetApp DataOps Toolkit を使用して画像認識用のニューラル ネットワークをトレーニングおよび展開する手順について説明します。これは、NetAppストレージを組み込んだトレーニング ジョブを示す例として提供することを目的としています。

### 前提条件

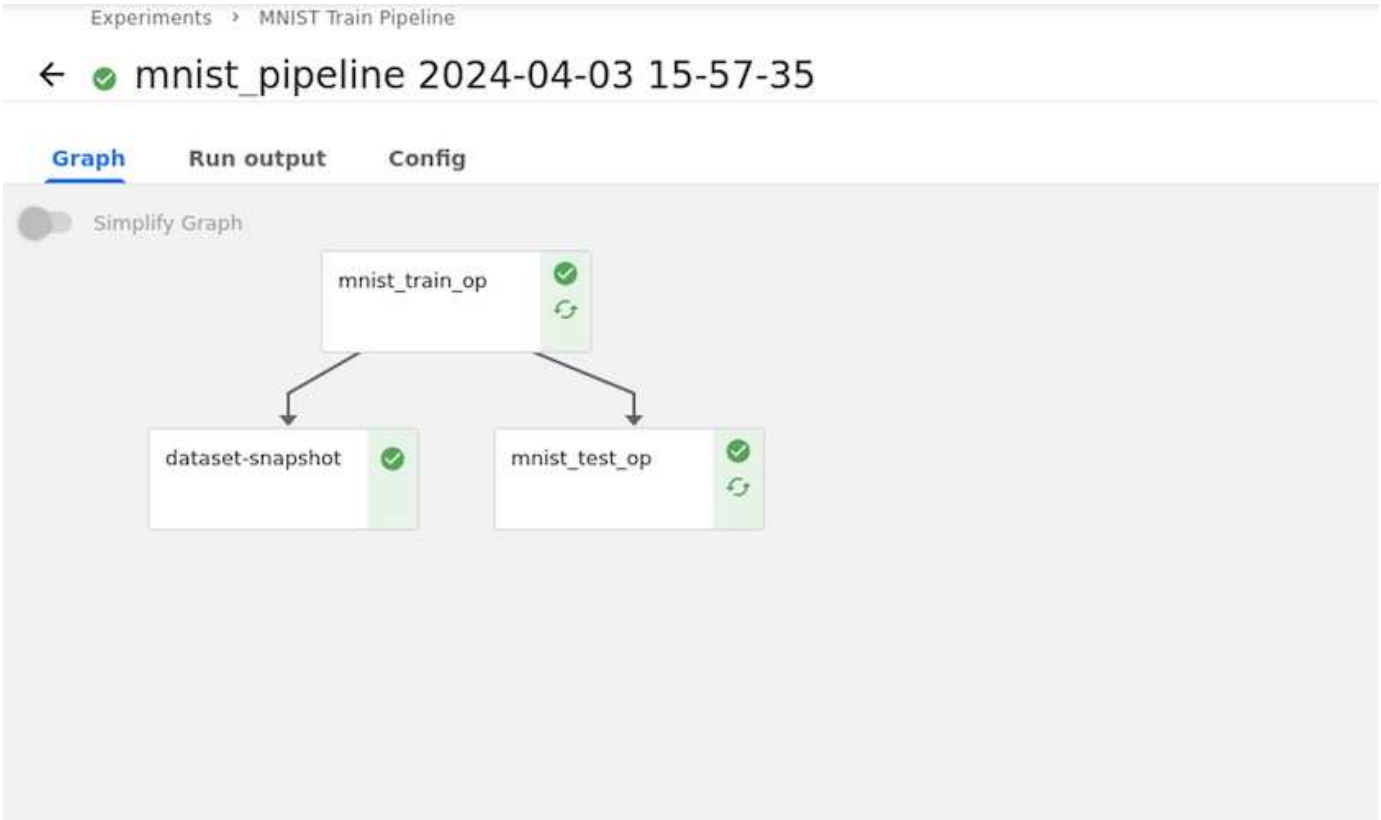
Kubeflow パイプライン内のトレーニングおよびテストのステップに使用するために必要な構成を含む Dockerfile を作成します。 Dockerfileの例は以下のとおりです。

```
FROM pytorch/pytorch:latest
RUN pip install torchvision numpy scikit-learn matplotlib tensorboard
WORKDIR /app
COPY . /app
COPY train_mnist.py /app/train_mnist.py
CMD ["python", "train_mnist.py"]
```

要件に応じて、プログラムの実行に必要なすべてのライブラリとパッケージをインストールします。機械学習モデルをトレーニングする前に、すでに機能している Kubeflow デプロイメントがあることを前提としています。

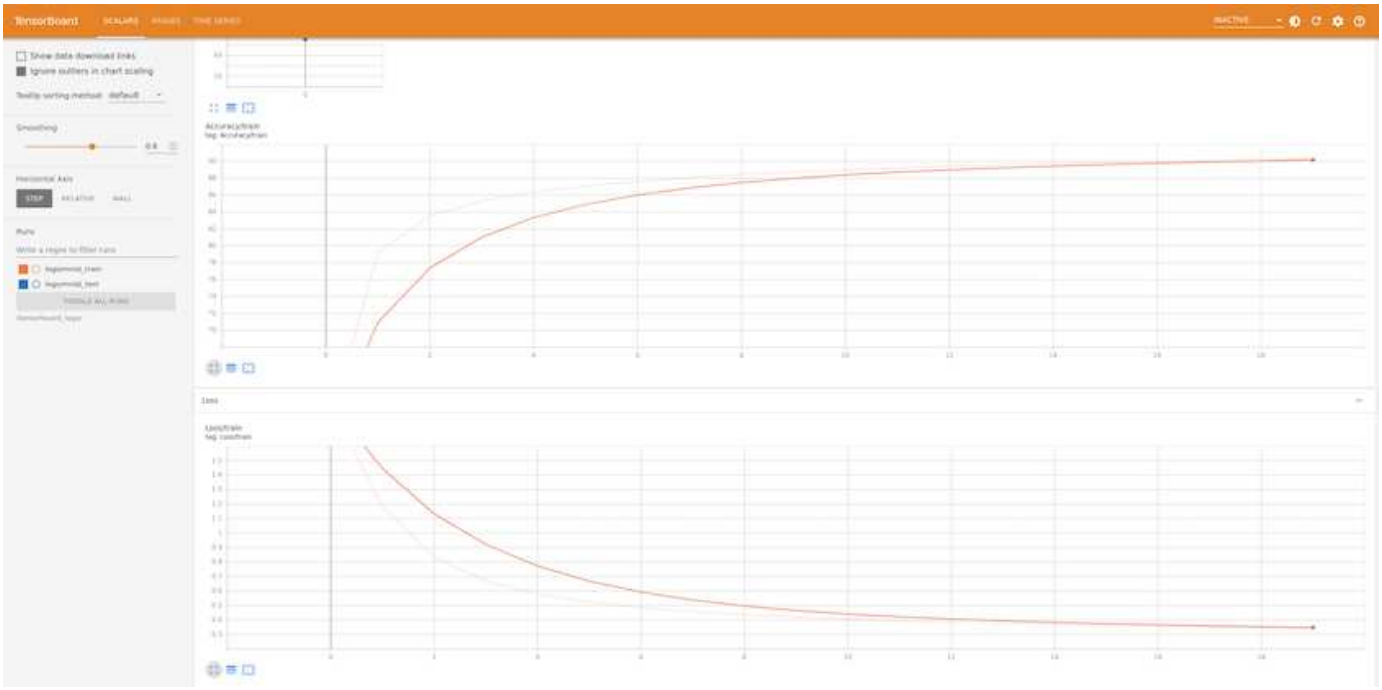
**PyTorch** と **Kubeflow** パイプラインを使用して **MNIST** データで小規模な **NN** をトレーニングする

MNIST データでトレーニングされた小さなニューラル ネットワークの例を使用します。 MNIST データセットは、0 ～ 9 の数字の手書き画像で構成されています。画像のサイズは 28x28 ピクセルです。データセットは、60,000 枚のトレーニング画像と 10,000 枚の検証画像に分かれています。この実験に使用されたニューラル ネットワークは 2 層のフィードフォワード ネットワークです。トレーニングは Kubeflow Pipelines を使用して実行されます。ドキュメントを参照してください "ここをクリックしてください。"詳細についてはこちらをご覧ください。 Kubeflow パイプラインには、前提条件セクションの Docker イメージが組み込まれています。



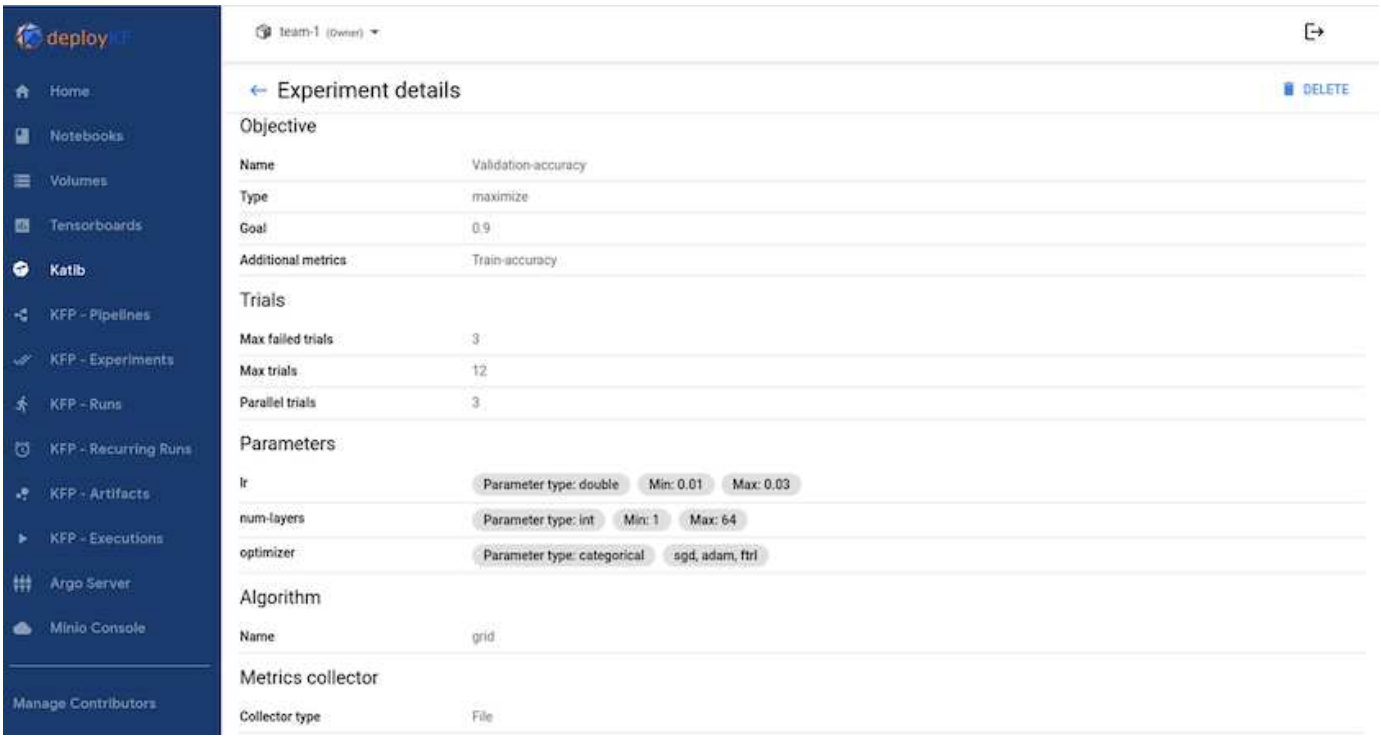

**Tensorboard** を使用して結果を視覚化する

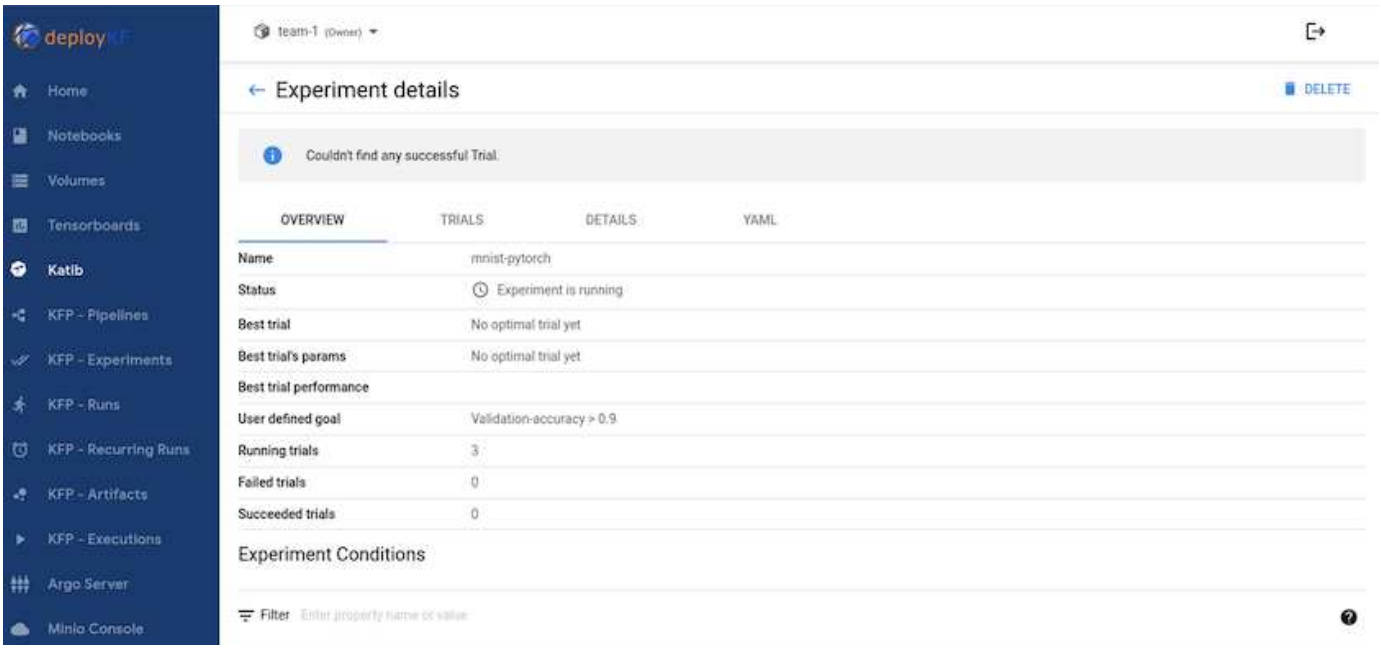
モデルをトレーニングしたら、Tensorboard を使用して結果を視覚化できます。 "テンソルボード" Kubeflow ダッシュボードの機能として利用できます。ジョブに合わせてカスタム テンソルボードを作成できます。以下の例は、トレーニング精度とエポック数、およびトレーニング損失とエポック数のプロットを示しています。



**Katib** を使用したハイパーパラメータの実験

"カティブ"は、モデルのハイパーパラメータを試すために使用できる Kubeflow 内のツールです。実験を作成するには、まず目的の指標/目標を定義します。これは通常、テストの精度です。メトリックが定義されたら、試してみたいハイパーパラメータ (オプティマイザー/学習率/レイヤー数) を選択します。 Katib は、ユーザー定義の値を使用してハイパーパラメータ スイープを実行し、目的のメトリックを満たす最適なパラメータの組み合わせを見つけます。これらのパラメータは UI の各セクションで定義できます。あるいは、必要な仕様を記述した **YAML** ファイルを定義することもできます。以下はカティブ実験の図解です。






**NetApp**スナップショットを使用して追跡可能なデータを保存する

モデルのトレーニング中に、追跡可能性のためにトレーニング データセットのスナップショットを保存する場合があります。これを実行するには、以下に示すように、パイプラインにスナップショット ステップを追加します。スナップショットを作成するには、 "Kubernetes 用NetApp DataOps ツールキット" 。

```
@dsl.pipeline(
    name = 'MNIST Classification Pipeline',
    description = 'Train a simple NN for classification'
)
def mnist_pipeline():
    mnist_train_task = mnist_train_op()
    mnist_train_task.apply(
        kfp.onprem.mount_pvc('mnist-data', 'mnist-data-vol', '/mnt/data/')
    )

    mnist_test_task = mnist_test_op()
    mnist_test_task.apply(
        kfp.onprem.mount_pvc('mnist-data', 'mnist-data-vol', '/mnt/data/')
    )

    volume_snapshot_name = "mnist-pytorch-snapshot"
    dataset_snapshot = dsl.ContainerOp(
        name="dataset-snapshot",
        image="python:3.9",
        command=["/bin/bash", "-c"],
        arguments=["\
            python3 -m pip install netapp-dataops-k8s && \
            echo '" + volume_snapshot_name + "' > /volume_snapshot_name.txt && \
            netapp_dataops_k8s_cli.py create volume-snapshot --pvc-name=" + "mnist-data" + " --snapshot-name=" + str(volume_snapshot_name) + " --namespace={{workflow.namespace}}"],
        file_outputs={"volume_snapshot_name": "/volume_snapshot_name.txt"}
    )
    mnist_test_task.after(mnist_train_task)
    dataset_snapshot.after(mnist_train_task)
```

参照 "Kubeflow 用のNetApp DataOps Toolkit の例"詳細についてはこちらをご覧ください。

# Trident操作の例

このセクションには、 Tridentで実行するさまざまな操作の例が含まれています。

## 既存のボリュームをインポートする

Kubernetes クラスター内のコンテナーにマウントするが、クラスター内の PVC に関連付けられていない既存のボリュームがNetAppストレージ システム/プラットフォーム上に存在する場合は、これらのボリュームをインポートする必要があります。これらのボリュームをインポートするには、 Tridentボリューム インポート機能を使用できます。

以下のコマンド例は、 `pb_fg_all` 。 PVCの詳細については、 "Kubernetesの公式ドキュメント" 。ボリュームインポート機能の詳細については、 "Tridentのドキュメント" 。

アン `accessModes`の価値 `ReadOnlyMany`サンプルの PVC 仕様ファイルで指定されています。詳細については、 `accessMode`フィールドについては、 "Kubernetesの公式ドキュメント" 。

```
$ cat << EOF > ./pvc-import-pb_fg_all-iface1.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all-iface1
  namespace: default
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain-iface1
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface1 pb_fg_all -f ./pvc-
import-pb_fg_all-iface1.yaml -n trident
+---------------------------------+--------
+-------------------------------------+----------
+-----------------------------------------------+--------+----------+
|          NAME                   |  SIZE  |       STORAGE CLASS
| PROTOCOL |             BACKEND UUID                          | STATE  |
MANAGED |
+---------------------------------+--------
+-------------------------------------+----------
+-----------------------------------------------+--------+----------+
| default-pb-fg-all-iface1-7d9f1 | 10 TiB | ontap-ai-flexgroups-retain-
iface1 | file     | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true
|
+---------------------------------+--------
+-------------------------------------+----------
+-----------------------------------------------+--------+----------+
$ tridentctl get volume -n trident
+-------------------------------------+---------
+-------------------------------------+----------
+---------------------------------------+--------+----------+
|               NAME                |  SIZE   |           STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+-------------------------------------+---------
+-------------------------------------+----------
+---------------------------------------+--------+----------+
| default-pb-fg-all-iface1-7d9f1   | 10 TiB  | ontap-ai-flexgroups-retain-
iface1 | file     | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true
|
```

```
+-----------------------------------+---------
+-----------------------------------+----------
+-----------------------------------------+--------+---------+
$ kubectl get pvc
NAME                  STATUS   VOLUME                                  CAPACITY
ACCESS MODES   STORAGECLASS                        AGE
pb-fg-all-iface1      Bound    default-pb-fg-all-iface1-7d9f1
10995116277760   ROX            ontap-ai-flexgroups-retain-iface1   25h
```

## 新しいボリュームをプロビジョニングする

Tridentを使用して、 NetAppストレージ システムまたはプラットフォームに新しいボリュームをプロビジョニングできます。

### kubectl を使用して新しいボリュームをプロビジョニングする

次のコマンド例は、kubectl を使用して新しいFlexVol volumeをプロビジョニングする方法を示しています。

アン `accessModes`の価値 `ReadWriteMany`次のサンプル PVC 定義ファイルで指定されています。詳細については、 `accessMode`フィールドについては、 "Kubernetesの公式ドキュメント" 。

```
$ cat << EOF > ./pvc-tensorflow-results.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tensorflow-results
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-ai-flexvols-retain
EOF
$ kubectl create -f ./pvc-tensorflow-results.yaml
persistentvolumeclaim/tensorflow-results created
$ kubectl get pvc
NAME                                STATUS    VOLUME
CAPACITY         ACCESS MODES   STORAGECLASS                          AGE
pb-fg-all-iface1                    Bound     default-pb-fg-all-iface1-7d9f1
10995116277760   ROX            ontap-ai-flexgroups-retain-iface1   26h
tensorflow-results                  Bound     default-tensorflow-results-
2fd60   1073741824       RWX            ontap-ai-flexvols-retain
25h
```

### NetApp DataOps ツールキットを使用して新しいボリュームをプロビジョニングする

NetApp DataOps Toolkit for Kubernetes を使用して、 NetAppストレージ システムまたはプラットフォームに新しいボリュームをプロビジョニングすることもできます。 NetApp DataOps Toolkit for Kubernetes は、Tridentを使用してボリュームをプロビジョニングしますが、ユーザーのプロセスを簡素化します。参照"ドキュメント"詳細については。

# AIPod展開の高パフォーマンスジョブの例

## 単一ノードのAIワークロードを実行する

Kubernetes クラスターで単一ノードの AI および ML ジョブを実行するには、デプロイメント ジャンプ ホストから次のタスクを実行します。 Tridentを使用すると、ペタバイト規模のデータを含む可能性のあるデータ ボリュームを、Kubernetes ワークロードから迅速かつ簡単にアクセスできるようになります。このようなデータ ボリュームを Kubernetes ポッド内からアクセスできるようにするには、ポッド定義で PVC を指定するだけです。

> このセクションでは、Kubernetes クラスターで実行しようとしている特定の AI および ML ワークロードがすでにコンテナー化 (Docker コンテナー形式) されていることを前提としています。

1. 次のコマンド例は、ImageNet データセットを使用する TensorFlow ベンチマーク ワークロード用の Kubernetes ジョブの作成を示しています。 ImageNetデータセットの詳細については、 "ImageNetウェブサイト" 。

   このサンプルジョブは 8 個の GPU を要求するため、8 個以上の GPU を備えた単一の GPU ワーカーノードで実行できます。このサンプルジョブは、8 個以上の GPU を搭載したワーカーノードが存在しない、または現在別のワークロードで使用されているクラスターで送信できます。その場合、そのようなワーカーノードが利用可能になるまで、ジョブは保留状態のままになります。

   さらに、ストレージ帯域幅を最大化するために、必要なトレーニング データを含むボリュームは、このジョブによって作成されるポッド内に 2 回マウントされます。ポッドには別のボリュームも搭載されています。この 2 番目のボリュームは、結果とメトリックを保存するために使用されます。これらのボリュームは、PVC の名前を使用してジョブ定義で参照されます。 Kubernetesジョブの詳細については、 "Kubernetesの公式ドキュメント" 。

   アン `emptyDir`ボリューム付き `medium`の価値 `Memory`に取り付けられている `/dev/shm`このサンプルジョブが作成するポッド内。デフォルトのサイズは `/dev/shm`Docker コンテナ ランタイムによって自動的に作成される仮想ボリュームは、TensorFlow のニーズを満たせない場合があります。マウント `emptyDir`次の例のようなボリュームは、十分な大きさの `/dev/shm`仮想ボリューム。詳細については `emptyDir`巻については、 "Kubernetesの公式ドキュメント" 。

   この例のジョブ定義で指定されている単一のコンテナには、 `securityContext > privileged`の価値 `true`。この値は、コンテナがホスト上で実質的にルートアクセス権を持っていることを意味します。この場合、実行される特定のワークロードにはルート アクセスが必要であるため、このアノテーションが使用されます。具体的には、ワークロードが実行するキャッシュクリア操作にはルート アクセスが必要です。これが `privileged: true`アノテーションが必要かどうかは、実行している特定のワークロードの要件によって異なります。

```
$ cat << EOF > ./netapp-tensorflow-single-imagenet.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-single-imagenet
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--dgx_version=dgx1", "--
num_devices=8"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-single-imagenet.yaml
job.batch/netapp-tensorflow-single-imagenet created
```

```
$ kubectl get jobs
NAME                                          COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet             0/1           24s        24s
```

2. 手順 1 で作成したジョブが正しく実行されていることを確認します。次のコマンド例は、ジョブ定義で指定されたとおりにジョブに対して単一のポッドが作成され、このポッドが現在 GPU ワーカー ノードの 1 つで実行されていることを確認します。

```
$ kubectl get pods -o wide
NAME                                                READY   STATUS
RESTARTS   AGE
IP               NODE             NOMINATED NODE
netapp-tensorflow-single-imagenet-m7x92             1/1     Running     0
3m    10.233.68.61    10.61.218.154   <none>
```

3. 手順 1 で作成したジョブが正常に完了したことを確認します。次のコマンド例は、ジョブが正常に完了したことを確認します。

```
$ kubectl get jobs
NAME                                                COMPLETIONS   DURATION
AGE
netapp-tensorflow-single-imagenet                   1/1           5m42s
10m
$ kubectl get pods
NAME                                                      READY   STATUS
RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92                   0/1     Completed
0          11m
$ kubectl logs netapp-tensorflow-single-imagenet-m7x92
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-
PERMISSIONS in file gds_dstore.c at line 702
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-
PERMISSIONS in file gds_dstore.c at line 711
Total images/sec = 6530.59125
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1 >
/proc/sys/vm/drop_caches'
==========================================
mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by
slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_be
nchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers=False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000
--datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu >
/tmp/20190814_105450_tensorflow_horovod_rdma_resnet50_gpu_8_256_b500_ima
genet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

4. オプション: ジョブの成果物をクリーンアップします。次のコマンド例は、手順 1 で作成されたジョブ オブジェクトの削除を示しています。

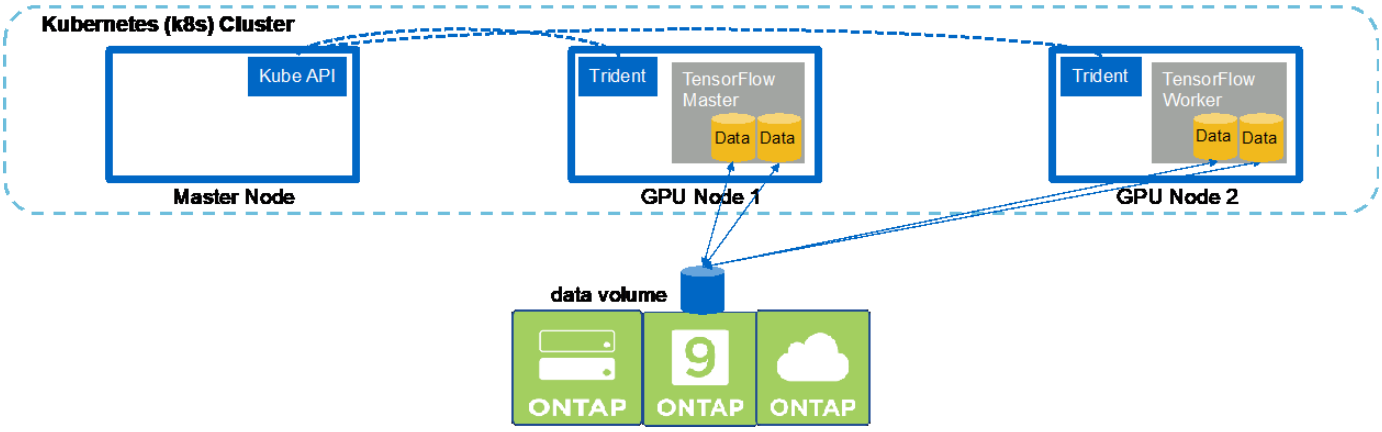
   ジョブ オブジェクトを削除すると、Kubernetes は関連付けられているポッドを自動的に削除します。

```
$ kubectl get jobs
NAME                                                COMPLETIONS   DURATION
AGE
netapp-tensorflow-single-imagenet                   1/1           5m42s
10m
$ kubectl get pods
NAME                                                      READY   STATUS
RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92                   0/1     Completed
0          11m
$ kubectl delete job netapp-tensorflow-single-imagenet
job.batch "netapp-tensorflow-single-imagenet" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

## 同期分散AIワークロードを実行する

Kubernetes クラスターで同期マルチノード AI および ML ジョブを実行するには、デプロイメント ジャンプ ホストで次のタスクを実行します。このプロセスにより、 NetApp ボリュームに保存されているデータを活用し、単一のワーカー ノードが提供できるよりも多くの GPU を使用できるようになります。同期分散 AI ジョブの図については、次の図を参照してください。

同期分散ジョブは、非同期分散ジョブと比較して、パフォーマンスとトレーニングの精度を向上させることができます。同期ジョブと非同期ジョブの長所と短所に関する説明は、このドキュメントの範囲外です。




1. 以下のコマンド例は、セクションの例で単一ノードで実行された同じTensorFlowベンチマークジョブの同期分散実行に参加する1つのワーカーの作成を示しています。"単一ノードのAIワークロードを実行する" 。この特定の例では、ジョブが 2 つのワーカー ノード間で実行されるため、1 つのワーカーのみがデプロイされます。

この例のワーカー デプロイメントでは 8 個の GPU が要求されるため、8 個以上の GPU を備えた単一の GPU ワーカー ノードで実行できます。 GPU ワーカー ノードに 8 個を超える GPU が搭載されている場合は、パフォーマンスを最大限に高めるために、この数をワーカー ノードに搭載されている GPU の数と同じになるように増やすことをおすすめします。 Kubernetesのデプロイメントの詳細については、"Kubernetesの公式ドキュメント" 。

この例では、この特定のコンテナ化されたワーカーは単独では完了しないため、Kubernetes デプロイメントが作成されます。したがって、Kubernetes ジョブ構造を使用してデプロイするのは意味がありません。ワーカーが単独で完了するように設計または記述されている場合は、ジョブ コンストラクトを使用してワーカーをデプロイすると効果的です。

この例のデプロイメント仕様で指定されているポッドには、 `hostNetwork`の価値 `true`。この値は、Kubernetes が通常各ポッドに対して作成する仮想ネットワーク スタックではなく、ホスト ワーカー ノードのネットワーク スタックをポッドが使用することを意味します。この場合、このアノテーションが使用されるのは、特定のワークロードが Open MPI、NCCL、および Horovod に依存して、ワークロードを同期分散方式で実行するためです。したがって、ホスト ネットワーク スタックへのアクセスが必要です。 Open MPI、NCCL、Horovod に関する説明は、このドキュメントの範囲外です。これが `hostNetwork: true`アノテーションが必要かどうかは、実行している特定のワークロードの要件によって異なります。詳細については、 `hostNetwork`フィールドについては、 "Kubernetesの公式ドキュメント" 。

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-worker.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapp-tensorflow-multi-imagenet-worker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapp-tensorflow-multi-imagenet-worker
  template:
    metadata:
      labels:
        app: netapp-tensorflow-multi-imagenet-worker
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
```

```
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["bash", "/netapp/scripts/start-slave-multi.sh",
"22122"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-worker.yaml
deployment.apps/netapp-tensorflow-multi-imagenet-worker created
$ kubectl get deployments
NAME                                        DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
1           4s
```

2. 手順 1 で作成したワーカー デプロイメントが正常に起動したことを確認します。次のコマンド例は、デプロイメント定義に示されているように、デプロイメントに対して単一のワーカー ポッドが作成され、このポッドが現在 GPU ワーカー ノードの 1 つで実行されていることを確認します。

```
$ kubectl get pods -o wide
NAME                                                              READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running   0          60s   10.61.218.154   10.61.218.154   <none>
$ kubectl logs netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725
22122
```

3. 同期マルチノード ジョブの実行を開始、参加、追跡するマスター用の Kubernetes ジョブを作成します。以下のコマンド例は、セクションの例で単一ノードで実行された同じTensorFlowベンチマークジョブの同期分散実行を開始、参加、追跡するマスターを1つ作成します。"単一ノードのAIワークロードを実行する"

。

この例のマスタージョブは 8 つの GPU を要求するため、8 個以上の GPU を備えた単一の GPU ワーカーノードで実行できます。 GPU ワーカー ノードに 8 個を超える GPU が搭載されている場合は、パフォーマンスを最大限に高めるために、この数をワーカー ノードに搭載されている GPU の数と同じになるように増やすことをおすすめします。

このジョブ定義例で指定されているマスターポッドには、 `hostNetwork`の価値 `true`労働者ポッドに与えられたのと同じように `hostNetwork`の価値 `true`ステップ1で。この値が必要な理由の詳細については、手順 1 を参照してください。

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-master.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-multi-imagenet-master
spec:
  backoffLimit: 5
  template:
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--port=22122", "--
num_devices=16", "--dgx_version=dgx1", "--
nodes=10.61.218.152,10.61.218.154"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
```

```
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-master.yaml
job.batch/netapp-tensorflow-multi-imagenet-master created
$ kubectl get jobs
NAME                                        COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   0/1           25s        25s
```

4. 手順 3 で作成したマスター ジョブが正しく実行されていることを確認します。次のコマンド例は、ジョブ定義に示されているように、ジョブに対して単一のマスター ポッドが作成され、このポッドが現在 GPU ワーカー ノードの 1 つで実行されていることを確認します。また、手順 1 で最初に確認したワーカー ポッドがまだ実行されており、マスター ポッドとワーカー ポッドが異なるノードで実行されていることも確認する必要があります。

```
$ kubectl get pods -o wide
NAME                                                              READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-multi-imagenet-master-ppwwj                     1/1
Running   0          45s   10.61.218.152   10.61.218.152   <none>
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running   0          26m   10.61.218.154   10.61.218.154   <none>
```

5. 手順 3 で作成したマスター ジョブが正常に完了したことを確認します。次のコマンド例は、ジョブが正常に完了したことを確認します。

```
$ kubectl get jobs
NAME                                        COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      9m18s
$ kubectl get pods
NAME                                                              READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj                     0/1
Completed   0          9m38s
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0          35m
$ kubectl logs netapp-tensorflow-multi-imagenet-master-ppwwj
```

```
[10.61.218.152:00008] WARNING: local probe returned unhandled
shell:unknown assuming bash
rm: cannot remove '/lib': Is a directory
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 702
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 702
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
Total images/sec = 12881.33875
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 2 -H 10.61.218.152:1,10.61.218.154:1 -mca
pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" bash -c 'sync; echo 1 >
/proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 16 -H 10.61.218.152:8,10.61.218.154:8
-bind-to none -map-by slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH
-mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -x
NCCL_IB_HCA=mlx5 -x NCCL_NET_GDR_READ=1 -x NCCL_IB_SL=3 -x
NCCL_IB_GID_INDEX=3 -x
NCCL_SOCKET_IFNAME=enp5s0.3091,enp12s0.3092,enp132s0.3093,enp139s0.3094
-x NCCL_IB_CUDA_SUPPORT=1 -mca orte_base_help_aggregate 0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_be
nchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers=False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000 --
datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu >
/tmp/20190814_161609_tensorflow_horovod_rdma_resnet50_gpu_16_256_b500_im
agenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

6. 不要になったらワーカーデプロイメントを削除します。次のコマンド例は、手順 1 で作成されたワーカーデプロイメント オブジェクトの削除を示しています。

   ワーカー デプロイメント オブジェクトを削除すると、Kubernetes は関連付けられているワーカー ポッドを自動的に削除します。

```
$ kubectl get deployments
NAME                                          DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
1           43m
$ kubectl get pods
NAME                                                              READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj                     0/1
Completed   0          17m
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0          43m
$ kubectl delete deployment netapp-tensorflow-multi-imagenet-worker
deployment.extensions "netapp-tensorflow-multi-imagenet-worker" deleted
$ kubectl get deployments
No resources found.
$ kubectl get pods
NAME                                               READY   STATUS
RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0
18m
```

7. オプション: マスタージョブアーティファクトをクリーンアップします。次のコマンド例は、手順 3 で作成されたマスター ジョブ オブジェクトの削除を示しています。

   マスタージョブオブジェクトを削除すると、Kubernetes は関連付けられているすべてのマスターポッドを自動的に削除します。

```
$ kubectl get jobs
NAME                                          COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      19m
$ kubectl get pods
NAME                                               READY   STATUS
RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0
19m
$ kubectl delete job netapp-tensorflow-multi-imagenet-master
job.batch "netapp-tensorflow-multi-imagenet-master" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

## 著作権に関する情報



## 商標に関する情報