NetAppによるオープンソースMLOps

NetApp Solutions

NetApp
May 10, 2024

This PDF was generated from https://docs.netapp.com/ja-jp/netapp-solutions/ai/aicp_introduction.html on May 10, 2024. Always check docs.netapp.com for the latest.

目次

NetAppによるオープンソースMLOps

NetAppによるオープンソースMLOps

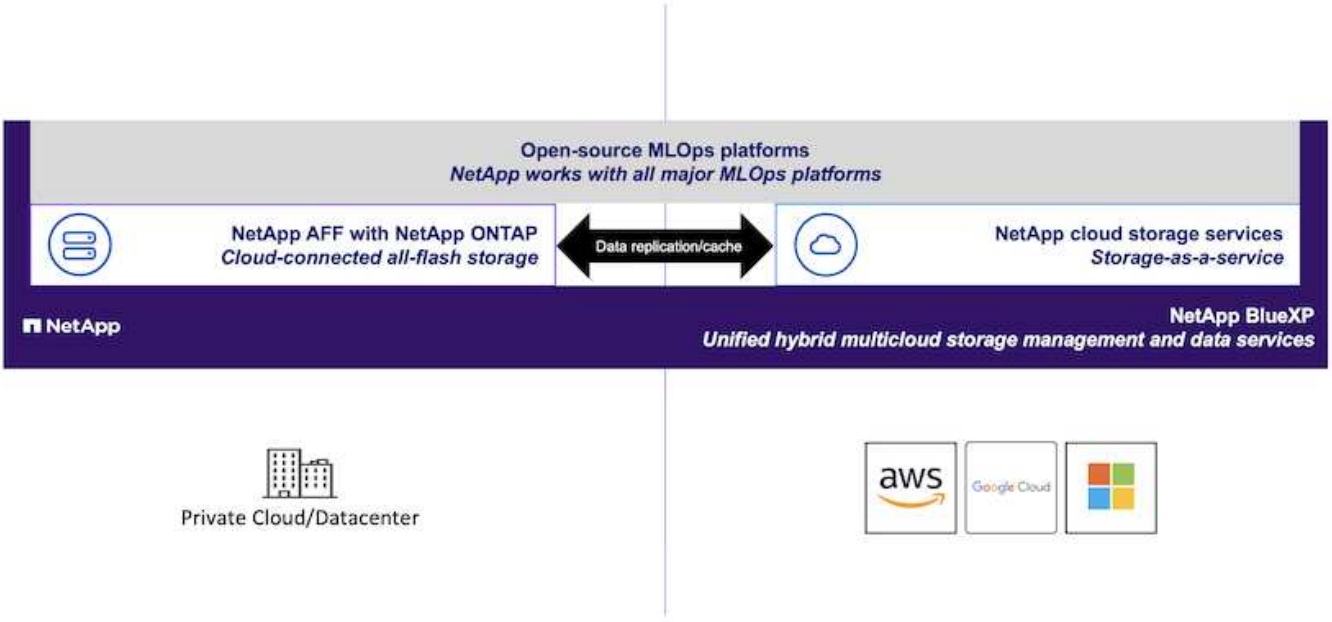
ネットアップ、 Mike Oglesby
NetApp、Mohan Acharya氏

あらゆる規模の企業や組織が、多くの業界で、人工知能（AI）、機械学習（ML）、ディープラーニング（DL）を導入して、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、競争が激化する市場で優位に立つことになりつつあります。AI 、 ML 、 DL の利用が増えるにつれ、ワークロードの拡張性やデータの可用性など、多くの課題に直面しています。この解決策では、NetAppのデータ管理機能と一般的なオープンソースのツールやフレームワークを組み合わせることで、これらの課題に対処する方法について説明します。

この解決策は、MLOpsワークフローに組み込むことができるさまざまなオープンソースのツールとフレームワークを紹介することを目的としています。これらのさまざまなツールやフレームワークは、要件やユースケースに応じて、一緒に使用することも、単独で使用することもできます。

この解決策では、次のツール/フレームワークについて説明します。

- "Apache の通気"
- "クビフロー"



テクノロジの概要

人工知能

AI とは、人間の心の認識機能を模倣するためにコンピュータが訓練されているコンピュータ科学分野で

す。AI 開発者は、人間に似た方法、または人間に比べて優れた方法で、コンピュータをトレーニングして問題を解決します。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に対応するために、 AI 、 ML 、 DL を導入する傾向に迫られています。次に例を示します。

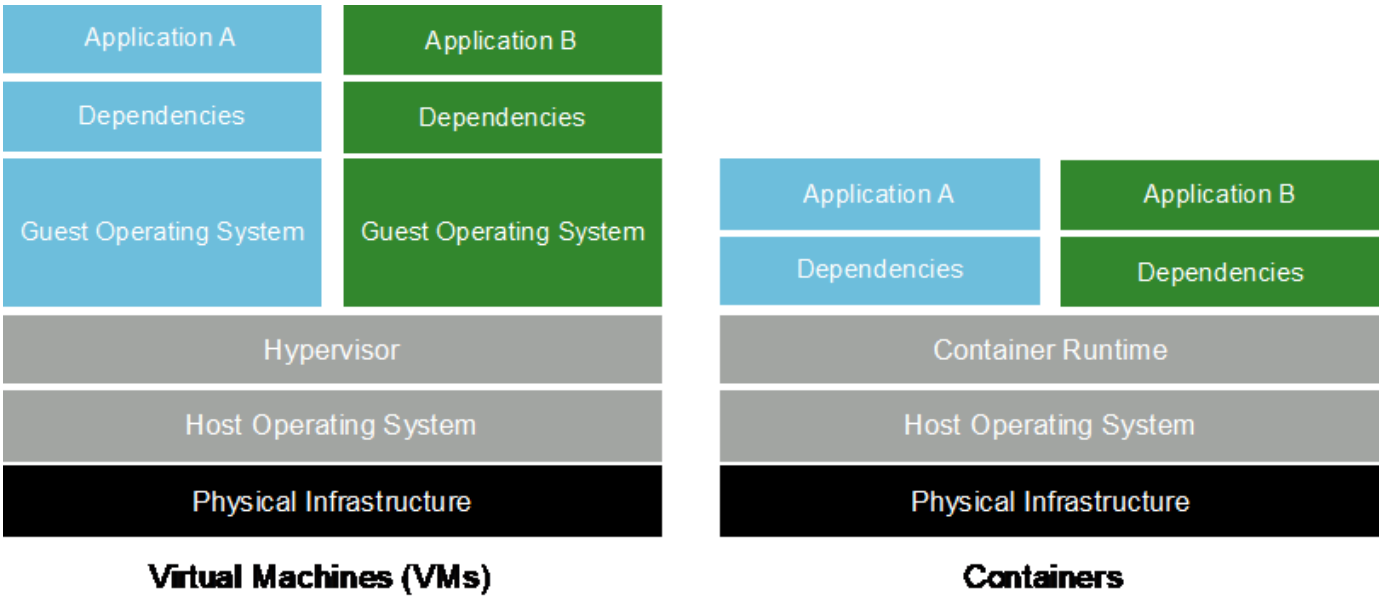
- 未知のビジネスに大量のデータを分析しています 分析
- 自然言語処理を使用して顧客と直接やり取りする
- さまざまなビジネスプロセスと機能を自動化します。

最新の AI トレーニングと推論のワークロードには、超並列処理機能が必要です。そのため、 GPU の並列処理機能は汎用 CPU よりもはるかに優れているため、 GPU を使用した AI 処理も増えています。

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独立したユーザスペースインスタンスです。コンテナの採用が急速に増加しています。コンテナは、仮想マシン（ VM ）が提供するものと同じアプリケーションのサンドボックス化のメリット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レイヤとゲストオペレーティングシステムレイヤが排除されているため、コンテナの軽量化が大幅に向上しています。次の図に、仮想マシンとコンテナを視覚的に示します。

コンテナを使用すると、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で直接効率的にパッケージングできます。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。これは、アプリケーションの依存関係がマシンに存在しない場合でも当てはまります。これは、すべての依存関係がコンテナ自体にパッケージ化されているためです。詳細については、を参照してください "Docker Web サイト"。



Kubernetes

Kubernetes は、 Google が当初設計した、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あり、 Cloud Native Computing Foundation （ CNCF ）によって管理されています。Kubernetes を使用すると、コンテナ化されたアプリケーションの導入、管理、拡張の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ています。詳細については、を参照してください "Kubernetes Web サイト"。

ネットアップアストラト **Trident**

Astra Tridentでは、パブリッククラウドやオンプレミスにあるONTAP（AFF、NetApp FAS、Select、Cloud、Amazon FSx for NetApp ONTAP）、Elementソフトウェア（NetApp HCI、SolidFire）、Azure NetApp Filesサービス、Cloud Volumes Service on Google CloudAstra Tridentは、Kubernetesとネイティブに統合される、コンテナストレージインターフェイス（CSI）に準拠した動的ストレージオーケストレーションツールです。

NetApp DataOps ツールキット

。 "NetApp DataOps ツールキット" は、ハイパフォーマンスなスケールアウト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の管理を簡易化する、Pythonベースのツールです。主な機能は次のとおりです。

- ハイパフォーマンスなスケールアウトNetAppストレージを基盤とする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
- 大容量ワークスペースのクローンをほぼ瞬時に作成し、実験や迅速な反復を可能にします。
- 大容量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、バックアップやトレーサビリティ/ベースライン化を実現します。
- ほぼインスタンス化することなく、大容量でハイパフォーマンスなデータボリュームをプロビジョニング、クローニング、およびSnapshot作成する。

クビフロー

Kubeflow は Kubernetes 向けのオープンソースの AI / ML ツールキットで、 Google が開発したものです。Kubeflow プロジェクトでは、 Kubernetes での AI ワークフローと ML ワークフローの導入を、シンプル、ポータブル、拡張性に優れた方法で実施します。KubeflowはKubernetesの複雑さを抽象化することで、データサイエンティストがデータサイエンスという自社が最もよく知っていることに集中できるようにします。表示については、次の図を参照してください。Kubeflowは、オールインワンMLOpsプラットフォームを希望する組織に適したオープンソースオプションです。詳細については、を参照してください "Kubeflow の Web サイト"。

Kubeflow パイプライン

Kubeflow Pipelines は Kubeflow の主要コンポーネントです。Kubeflow Pipelines は、移植性と拡張性に優れた AI および ML ワークフローを定義、導入するためのプラットフォームと標準です。詳細については、を参照してください "Kubeflow の公式ドキュメント"。

Jupyter Notebook Server の **2** つのツールを使用

Jupyter Notebook Server はオープンソースの Web アプリケーションで、データサイエンティストは Jupyter Notebook と呼ばれる Wiki 形式のドキュメントを作成できます。このドキュメントには、ライブコードと説明的なテストが含まれています。Jupyter Notebook は、 AI プロジェクトと ML プロジェクトを文書化、保存、共有する手段として、 AI と ML のコミュニティで広く使用されています。Kubeflow を使用すると、 Kubernetes での Jupyter Notebook Server のプロビジョニングと導入が簡単になります。Jupyter Notebook の詳細については、を参照してください "Jupyter のウェブサイト"。Kubeflow のコンテキスト内の Jupyter Notebook の詳細については、を参照してください "Kubeflow の公式ドキュメント"。

カティブ

Katibは、自動機械学習（AutoML）向けの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チューニング、早期停止、ニューラルアーキテクチャ検索（NAS）をサポートしている。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でサポートします。 その他。Katibは、Bayesian最適化、Tree of Parzen Estimators、Random Search、Covariance Matrix Adaptation Evolution Strategy、Hyperband、Efficient Neural Architecture Search、Differentiable Architecture Search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 のコンテキスト内の Jupyter Notebook の詳細については、を参照してください "Kubeflow の公式ドキュメント"。

Apache の通気

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オーサリング、スケジューリング、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。ETL やデータパイプラインのワークフローを自動化する目的でよく使用されますが、こうした種類のワークフローに限定されるわけではありません。Airflow プロジェクトは Airbnb が開始しましたが、業界で非常に人気があり、現在は Apache Software Foundation の後援を受けています。空気の流れは Python で書かれており、 Python スクリプトを使用して空気の流れが作られています。また、空気の流れは、「コードとしての設定」という原則に基づいて設計されています。 現在、多くの企業のエアフローユーザが Kubernetes の上で通気を実行しています。

ダイレクト非周期グラフ（ **DAG** ）

エアーフローでは、ワークフローは Directed Acyclic Graphs （ DAG ）と呼ばれます。DAG は、 DAG の定義に応じて、順番に実行されるタスク、並列タスク、またはその組み合わせで実行されるタスクで構成されます。エアーフロースケジューラは、 DAG 定義で指定されているタスクレベルの依存関係を維持しながら、一連のワーカーに対して個々のタスクを実行します。DAG は Python スクリプトを使用して定義および作成されます。

NetApp ONTAP

ネットアップが提供する最新世代のストレージ管理ソフトウェアONTAP 9を使用すれば、インフラを最新化し、クラウド対応のデータセンターに移行できます。ONTAP は、業界をリードするデータ管理機能を活用して、データの格納場所に関係なく、単一のツールセットでデータの管理と保護を実現します。エッジ、コア、クラウドなど、必要な場所に自由にデータを移動することもできます。ONTAP 9には、データ管理の簡易化、重要なデータの高速化と保護、ハイブリッドクラウドアーキテクチャ全体で次世代インフラ機能を実現する多数の機能が搭載されています。

データ管理を簡易化

データ管理は、AIアプリケーションの運用やAI / MLデータセットのトレーニングに適切なリソースを使用できるように、エンタープライズ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。以下に記載するネットアップテクノロジに関する追加情報 は、この検証の対象外ですが、導入環境によっては関連性がある場合もあります。

ONTAP データ管理ソフトウェアには、運用を合理化および簡易化し、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。

- インラインデータコンパクション、強化された重複排除：データコンパクションはストレージブロック内の無駄なスペースを削減し、重複排除は実効容量を大幅に増やします。この環境データはローカルに格納され、データはクラウドに階層化されます。
- 最小、最大、アダプティブのQuality of Service（AQoS）。きめ細かいサービス品質（QoS）管理機能により、高度に共有された環境で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。
- NetApp FabricPool の略。Amazon Web Services（AWS）、Azure、NetApp StorageGRID ストレージ解決策 など、パブリッククラウドとプライベートクラウドのストレージオプションへコールドデータを自動

的に階層化します。FabricPool の詳細については、を参照してください "TR-4598：『FabricPool best bests』"。

データの高速化と保護

ONTAP は、卓越したパフォーマンスとデータ保護を実現し、以下の方法でこれらの機能を拡張します。

- パフォーマンスとレイテンシの低下：ONTAP は、可能なかぎり最小の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 には、組み込みのデータ保護機能が用意されており、すべてのプラットフォームを共通の管理機能で管理できます。
- NetApp Volume Encryption（NVE）：ONTAP は、オンボードと外部キー管理の両方をサポートし、ボリュームレベルでのネイティブな暗号化を実現します。
- マルチテナンシーおよび多要素認証ONTAP を使用すると、最高レベルのセキュリティでインフラリソースを共有できます。

将来のニーズにも対応できるインフラ

ONTAP は、次の機能を備えており、要件が厳しく、絶えず変化するビジネスニーズに対応できます。

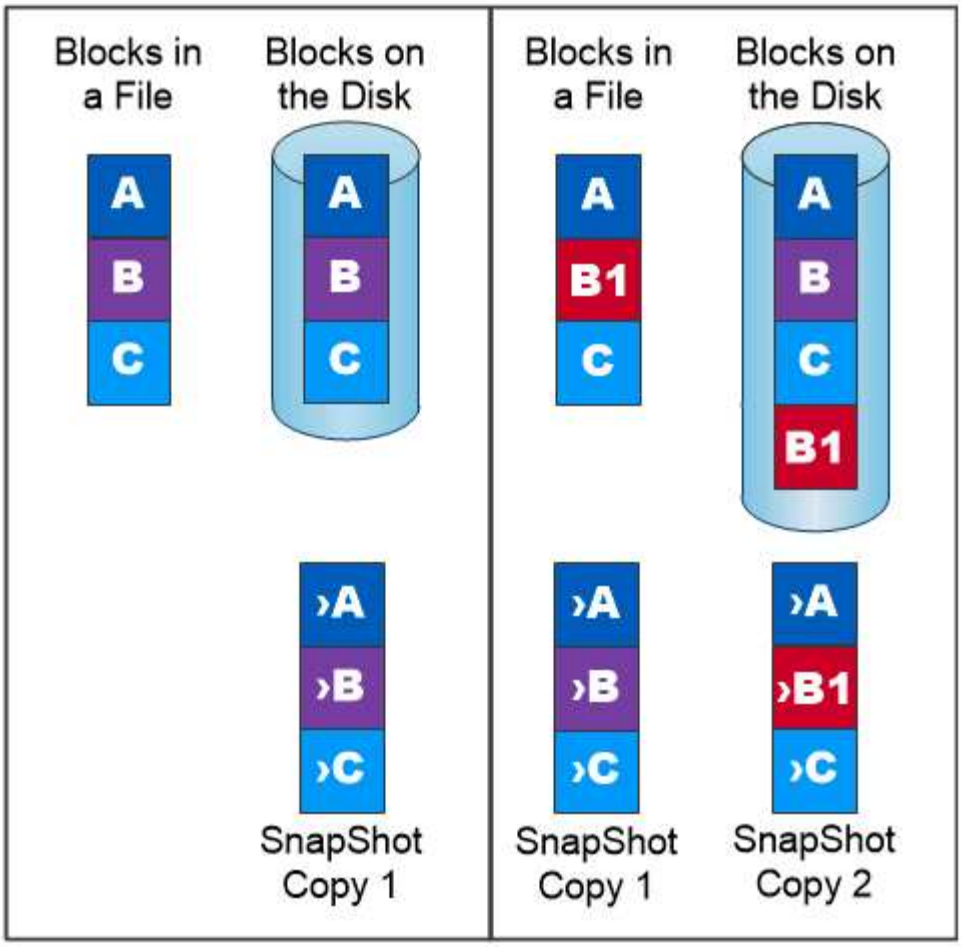
- シームレスな拡張とノンストップオペレーションONTAP を使用すると、既存のコントローラとスケールアウトクラスタに無停止で容量を追加できます。NVMe や 32Gb FC などの最新テクノロジへのアップグレードも、コストのかかるデータ移行やシステム停止を行わずに実行できます。
- クラウドへの接続：ONTAPは、ほとんどのクラウドに対応したストレージ管理ソフトウェアで、すべてのパブリッククラウドでSoftware-Defined Storageとクラウドネイティブインスタンスを選択できます。
- 新しいアプリケーションとの統合：ONTAP は、既存のエンタープライズアプリケーションをサポートするインフラを使用して、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やアプリケーション向けにエンタープライズクラスのデータサービスを提供します。

NetApp Snapshot コピー

NetApp Snapshot コピーは、ボリュームの読み取り専用のポイントインタイムイメージです。次の図に示すように、イメージには Snapshot コピーが最後に作成されたあとに作成されたファイルへの変更だけが記録されるため、ストレージスペースは最小限しか消費せず、パフォーマンスのオーバーヘッドもわずかです。

Snapshot コピーの効率性は、 ONTAP の中核的なストレージ仮想化テクノロジである Write Anywhere File Layout （ WAFL ）によって実現します。WAFL は、データベースと同様に、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、 WAFL は既存のブロックを上書きしません。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 では、 Snapshot コピーの作成時にデータブロックをコピーするのではなくメタデータを参照するため、非常に効率的です。他のシステムと違ってコピーするブロックを探すシーク時間もなければ、コピー自体を作成するコストもかかりません。

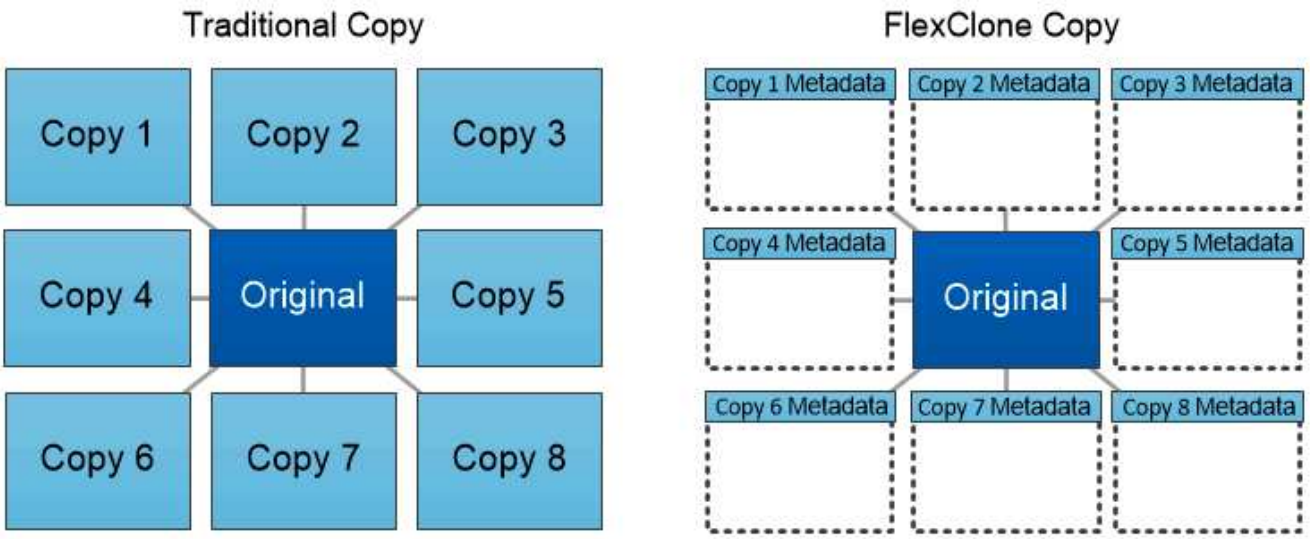
Snapshot コピーを使用して、個々のファイルまたは LUN をリカバリしたり、ボリュームの内容全体をリストアしたりできます。ONTAP は、 Snapshot コピーのポインタ情報をディスク上のデータと比較することで、ダウンタイムや多大なパフォーマンスコストなしで損失オブジェクトや破損オブジェクトを再構築します。



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 テクノロジは、 Snapshot 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。コピーと親でデータブロックが共有されるため、次の図に示すように、コピーに変更が書き込まれるまではメタデータに必要な分しかストレージは消費されません。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、同じデータセットのコピーが複数必要な状況（開発用ワークスペースなど）や一時的にデータセットのコピーが必要な状況（本番環境のデータセットでアプリケーションをテストする場合など）に適しています。



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

NetApp SnapMirror ソフトウェアは、データファブリック全体にわたる、コスト効率に優れた使いやすいユニファイドレプリケーション解決策です。LAN または WAN 経由でデータを高速で複製します。仮想環境と従来の環境の両方でビジネスクリティカルなアプリケーションを含む、あらゆるタイプのアプリケーションに対し、高いデータ可用性と高速なデータレプリケーションを提供します。1 つ以上のネットアップストレージシステムにデータをレプリケートし、セカンダリデータを継続的に更新すると、データが最新の状態に保たれ、必要なときにいつでも使用できます。外部レプリケーションサーバは必要ありません。SnapMirror テクノロジを利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 の Storage Efficiency 機能を活用します。SnapMirror ソフトウェアには、組み込みのネットワーク圧縮機能も使用して、データ転送を高速化し、ネットワーク帯域幅の使用量を最大 70% 削減します。SnapMirror テクノロジを使用すると、 1 つのシンレプリケーションデータストリームを利用して単一のリポジトリを作成し、アクティブな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できるため、ネットワークトラフィックを最大 50% 削減できます。

NetApp BlueXPのコピーと同期

BlueXPのコピーと同期は、迅速かつセキュア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、NetApp Cloud Volumes Service、Azure NetApp Files、AWS S3、AWS EFS、Azure Blob）間でファイルを転送する必要があるかどうか Google Cloud Storage（IBM Cloud Object Storage）のBlueXP Copy and Syncは、必要な場所に迅速かつ安全にファイルを移動します。

転送されたデータは、ソースとターゲットの両方で完全に使用できます。BlueXPの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データを継続的に同期したりできます。いずれにせよ、BlueXPのCopy and Syncは差分のみを移動するため、データレプリケーションにかかる時間とコストを最小限に抑えることができます。

BlueXPのCopy and Syncは、セットアップと使用が非常に簡単なソフトウェアサービス（SaaS）ツールです。BlueXPのCopyと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のCopy and Syncデータブローカーは、AWS、Azure、Google Cloud Platform、オンプレミスに導入できます。

NetApp XCP

NetApp XCP は、ネットアップとネットアップ間のデータ移行およびファイルシステムに関する分析情報を提供するクライアントベースのソフトウェアです。XCP は、大量のデータセットとハイパフォーマンスな移行を処理するために、利用可能なすべてのシステムリソースを活用することで、最大限のパフォーマンスを実現するように設計されています。ファイルシステムを完全に可視化するために XCP を使用すると、レポート生成オプションが利用できます。

NetApp XCP は、 NFS プロトコルと SMB プロトコルをサポートする単一パッケージで提供されます。NFS データセット用の Linux バイナリと SMB データセット用の Windows 実行可能ファイルが XCP に含まれています。

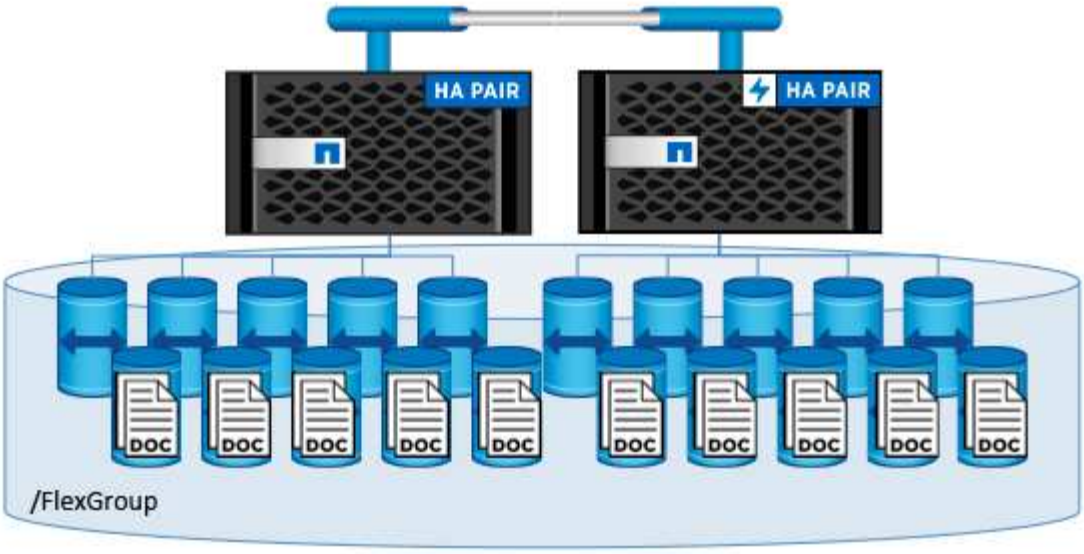
NetApp XCP File Analytics は、ファイル共有を検出し、ファイルシステム上でスキャンを実行し、ファイル分析用のダッシュボードを提供するホストベースのソフトウェアです。XCP File Analytics は、ネットアップシステムと他社システムの両方に対応し、 Linux ホストまたは Windows ホストで動作して、 NFS および SMB エクスポートファイルシステムの分析を提供します。

NetApp ONTAP FlexGroup Volume の略

トレーニングデータセットは、数十億に及ぶ可能性のあるファイルの集まりです。ファイルには、テキスト、オーディオ、ビデオなどの形式の非構造化データを含めることができます。これらのデータは、並行して読み込まれるように保存して処理する必要があります。ストレージシステムは、多数の小さなファイルを格納し、シーケンシャル I/O とランダム I/O でそれらのファイルを並行して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、複数のコンスティチュエントメンバーボリュームで構成される単一のネームスペースです。ストレージ管理者の視点で見ると、 FlexGroup ボリュームは管理され、 NetApp FlexVol ボリュームのように機能します。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、複数のボリュームやノードにまたがってストライプされることはありません。次の機能が有効になります。

- FlexGroup ボリュームは、数ペタバイトの容量と、メタデータ比率の高いワークロード向けの予測可能な低レイテンシを提供します。
- 同じネームスペースで最大 4 、 000 億個のファイルをサポートします。
- CPU 、ノード、アグリゲート、コンスティチュエント FlexVol ボリューム全体で NAS ワークロードの並列処理をサポートします。



アーキテクチャ

この解決策は特定のハードウェアに依存しません。解決策は、 Trident でサポートされている、任意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、クラウドサービスと互換性があります。たとえば、NetApp AFFストレージシステム、Amazon FSx for NetApp ONTAP、Azure NetApp Files、NetApp Cloud Volumes ONTAPインスタンスなどです。さらに、使用しているKubernetesバージョンがKubeflowとNetApp Astra Tridentでサポートされていれば、解決策を任意のKubernetesクラスタに実装できます。Kubeflow でサポートされる Kubernetes バージョンの一覧については、を参照してください "Kubeflow の公式ドキュメント"。Trident で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、を参照してください "Trident のドキュメント"。解決策の検証に使用した環境の詳細については、次の表を参照してください。

ソフトウェアコンポーネント	バージョン
Apache の通気	2.0.1
Apache Airflow Helm チャート	8.0.8
クビフロー	1.7、次の方法で導入 "deployKF" 0.1.1
Kubernetes	1.26
ネットアップアストラト Trident	23.07

サポート

NetAppは、Apache Airflow、Kubeflow、Kubernetesの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ているMLOpsプラットフォームに関心がある場合は、 "ネットアップにお問い合わせください" 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 Astra Tridentの設定

NetApp AIPod環境でのAstra Tridentバックエンドの例

Astra Tridentを使用してKubernetesクラスタ内のストレージリソースを動的にプロビジョニングするには、Tridentバックエンドを1つ以上作成する必要があります。次の例は、この解決策のコンポーネントを "NetApp AIPod"。バックエンドの詳細については、を参照してください "Astra Trident のドキュメント"。

1. NetAppでは、AIPod用にFlexGroup対応のTridentバックエンドを作成することを推奨しています。

   以降のコマンド例では、AIPod Storage Virtual Machine（SVM）用にFlexGroup対応のTridentバックエンドを作成しています。このバックエンドは、 `ontap-nas-flexgroup` ストレージドライバ。ONTAP では、 FlexVol と FlexGroup の 2 つの主要なデータボリュームタイプがサポートされます。FlexVol ボリュームのサイズは限られています（現時点では、最大サイズは環境によって異なります）。一方、FlexGroup ボリュームは最大 20PB 、 4 、 000 億ファイルまでリニアに拡張でき、データ管理を大幅に簡易化する単一のネームスペースを提供します。そのため、 FlexGroup ボリュームは、大量のデータに依存する AI や ML のワークロードに最適です。

   少量のデータを処理していて、 FlexGroup ボリュームではなく FlexVol ボリュームを使用する場合は、「 ONTAP-NAS-flexgroup 」ストレージドライバの代わりに「 ONTAP-NAS' ストレージドライバ」を使用する Trident バックエンドを作成できます。

```
$ 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では、FlexVol対応のTridentバックエンドを作成することも推奨しています。FlexVolボリュームは、永続的アプリケーションのホスト、結果の格納、出力、デバッグ情報などに使用できます。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。以降のコマンド例では、FlexVol対応のTridentバックエンドを1つ作成しています。

```
$ 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環境向けのKubernetesストレージクラスの例

Astra Tridentを使用してKubernetesクラスタ内のストレージリソースを動的にプロビジョニングするには、Kubernetes StorageClassを1つ以上作成する必要があります。次の例は、この解決策のコンポーネントをに導入する場合に作成するさまざまなタイプのStorageClassesを示しています。 "NetApp AIPod"。StorageClasses の詳細について

は、を参照してください "Astra Trident のドキュメント"。

1. NetAppでは、セクションで作成したFlexGroup対応のTridentバックエンド用にストレージクラスを作成することを推奨します。 "NetApp AIPod環境でのAstra Tridentバックエンドの例"、手順 1.以降のコマンド例では、セクションで作成した2つのバックエンドの例に対応する複数のStorageClassesを作成しています。 "NetApp AIPod環境でのAstra Tridentバックエンドの例"、ステップ1 - "RDMA経由のNFS" そうではありません

   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するため、次の例では「 Retain 」の「 ReclaimPolicy 」の値を使用しています。「 ReclaimPolicy 」フィールドの詳細については、公式を参照してください "Kubernetes のドキュメント"。

   注：次の例のStorageClassesで使用される最大転送サイズは262144です。この最大転送サイズを使用するには、それに応じてONTAPシステムの最大転送サイズを設定する必要があります。を参照してください "ONTAP のドキュメント" を参照してください。

   注：RDMA経由のNFSを使用するには、ONTAPシステムでRDMA経由のNFSを設定する必要があります。詳細については、https://docs.netapp.com/us-en/ontap/nfs-rdma/[ONTAPドキュメントのリンクを参照してください。

   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せん。

```
$ 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. に対応するストレージクラスを作成することも推奨します セクションで作成した FlexVol 対応の Trident バックエンド "AIPod環境向けのAstra Tridentバックエンドの例"、ステップ 2 。以下のコマンド例は、FlexVol ボリューム用の単一のストレージクラス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バックエンドが指定されていません。Kubernetesを使用してこのStorageClassを使用するボリュームを管理すると、Tridentは、`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
```

クビフロー

Kubeflow の導入

このセクションでは、 Kubernetes クラスタに Kubeflow を導入するために実行する必要の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導入の演習を行う前に、次の作業をすでに実行していることを前提としています。

1. すでに稼働中のKubernetesクラスタがあり、導入する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る。サポートされているKubernetesバージョンのリストについては、 "Kubeflow の公式ドキュメント"。
2. KubernetesクラスタにNetApp Astra Tridentをインストールして設定しておきます。Astra Tridentの詳細については、 "Astra Trident のドキュメント"。

デフォルトの **Kubernetes StorageClass** を設定します

Kubeflowを導入する前に、Kubernetesクラスタ内でデフォルトのStorageClassを指定することを推奨します。Kubeflowの導入プロセスで、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ことがあります。StorageClassがデフォルトのStorageClassとして指定されていない場合、導入に失敗する可能性があります。クラスタ内のデフォルトの StorageClass を指定するには、導入ジャンプホストから次のタスクを実行します。クラスタ内ですでにデフォルトの StorageClass を指定している場合は、この手順を省略できます。

1. 既存のストレージクラスの 1 つをデフォルトのストレージクラスとして指定します。次のコマンド例は、という名前のStorageClassの指定を示しています。 `ontap-ai-flexvols-retain` をデフォルト

のStorageClassとして設定します。

> 「 ONTAP-NAS-flexgroup 」の Trident バックエンドタイプには、かなり大きな最小 PVC サイズがあります。デフォルトでは、 Kubeflow はサイズが数 GB しかない PVC のプロビジョニングを試みます。したがって、 Kubeflow の導入の目的で、「 ONTAP-NAS-flexgroup 」バックエンドタイプをデフォルトの StorageClass として使用する StorageClass を指定しないでください。

```
$ 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の導入オプション

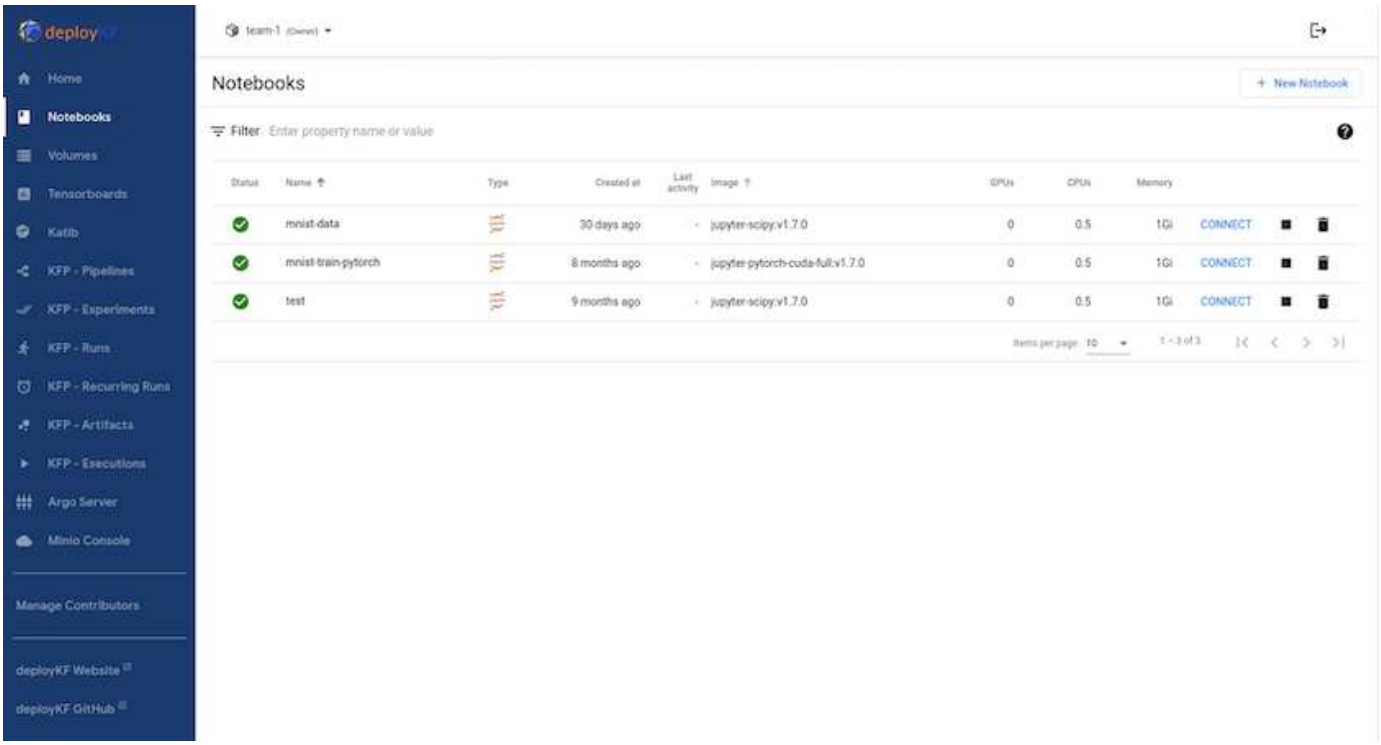
Kubeflowを導入するには、さまざまなオプションがあります。を参照してください "Kubeflow の公式ドキュメント" を参照し、ニーズに最適な導入オプションを選択してください。

> 検証のために、次のツールを使用してKubeflow 1.7を導入しました。 "deployKF" 0.1.1。

例： **Kubeflow** の操作とタスク

データサイエンティストまたは開発者向けに **Jupyter Notebook Workspace** をプロビジョニングします 使用

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 サーバの迅速なプロビジョニングを可能にします。Kubeflow コンテキスト内の Jupyter Notebook の詳細については、を参照してください "Kubeflow の公式ドキュメント"。



Kubeflowでの**NetApp DataOps Toolkit**の使用

。 "NetApp Data Science Toolkit for Kubernetes" Kubeflow と組み合わせて使用できます。NetApp Data Science Toolkit と Kubeflow を使用すると、次のようなメリットがあります。

- データサイエンティストは、NetAppの高度なデータ管理操作（スナップショットやクローンの作成など）を、Jupyterノートブックから直接実行できます。
- Kubeflow Pipelinesフレームワークを使用すると、スナップショットやクローンの作成など、NetAppの高度なデータ管理操作を自動化されたワークフローに組み込むことができます。

を参照してください "Kubeflow の例" ツールキットと Kubeflow を使用する場合の詳細については、 NetApp Data Science Toolkit GitHub リポジトリ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**ツールキットを使用した画像認識モデルのトレーニング

このセクションでは、KubeflowとNetApp DataOps Toolkitを使用した画像認識のためのニューラルネットワークのトレーニングと導入に関連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の例を示すことを目的としています。

前提条件

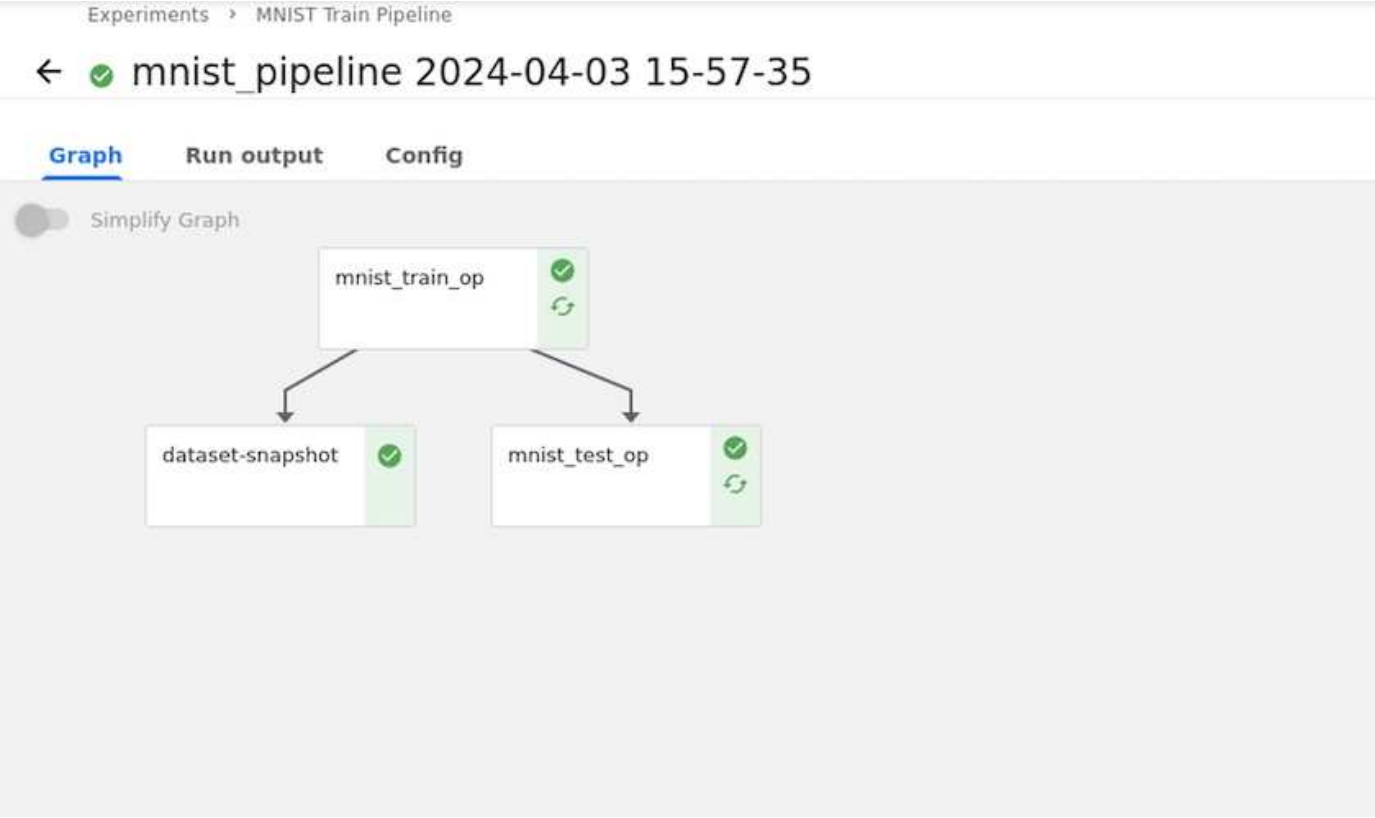
Kubeflowパイプライン内のトレーニングおよびテストステップに使用するために必要な構成を含むDockerfileを作成します。
Dockerfileの例を次に示します。

```
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"]
```

必要に応じて、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、すでに稼働しているKubeflowデプロイメントがあることを前提としています。

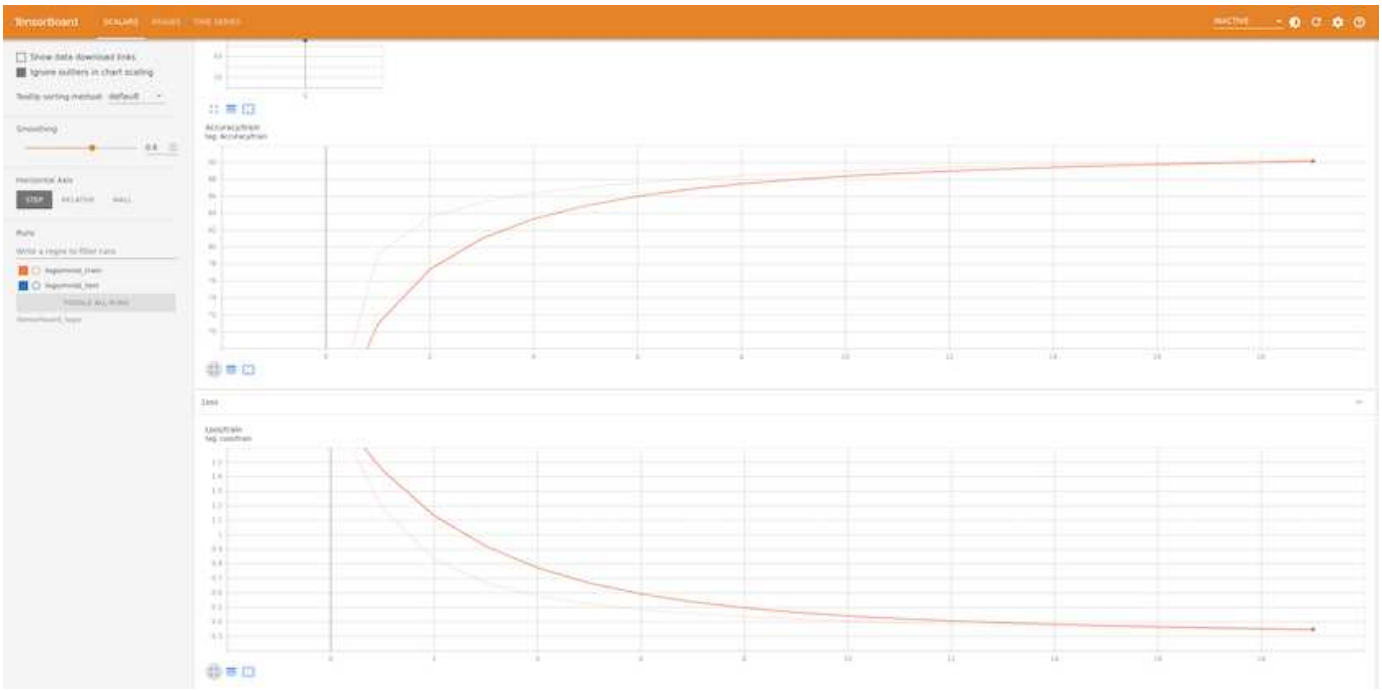
PyTorchと**Kubeflow**パイプラインを使用して、小規模**NN**を**MNIST**データでトレーニング

ここでは、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から9まで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枚の列車画像と10,000枚の検証画像に分割されています。この実験で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。のドキュメントを参照してください "こちらをご覧ください" を参照してください。Kubeflowパイプラインには、PrerequisitesセクションのDockerイメージが組み込まれています。



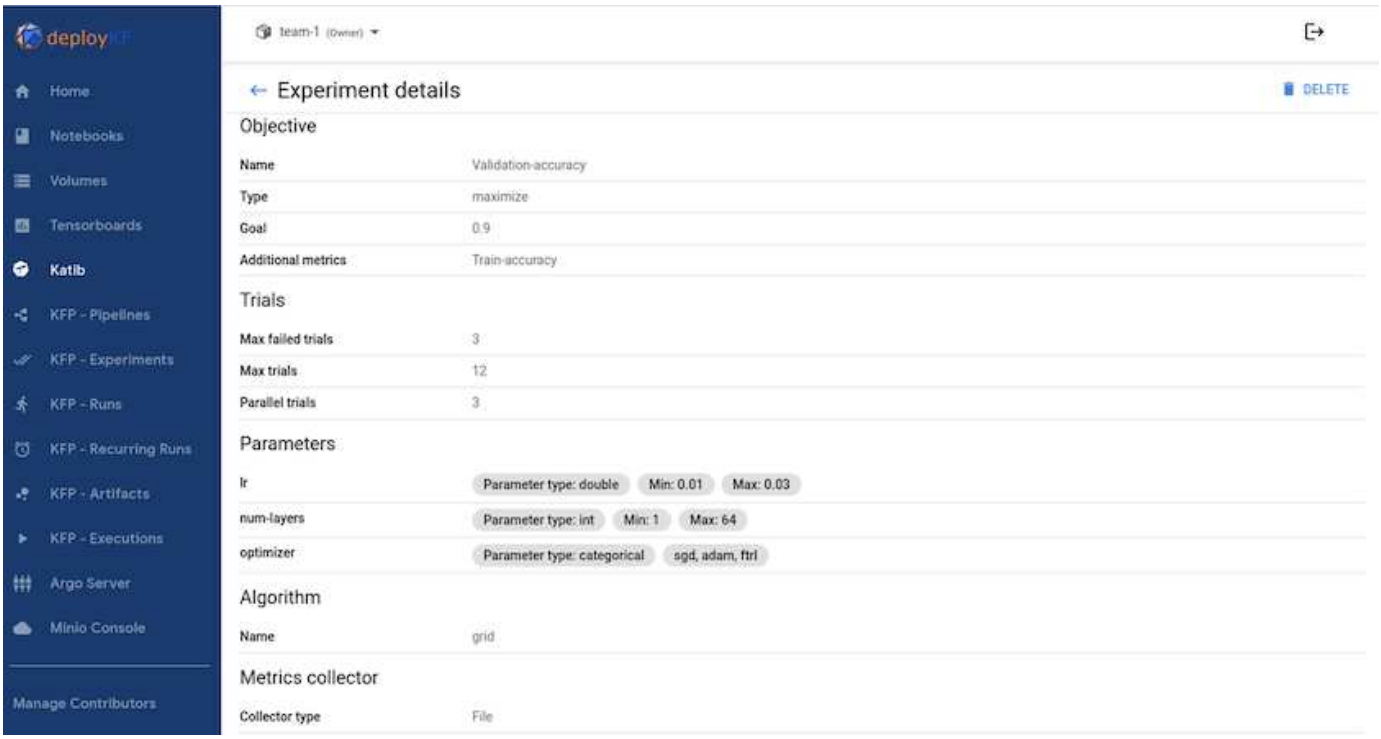
Tensorboardを使用した結果の表示

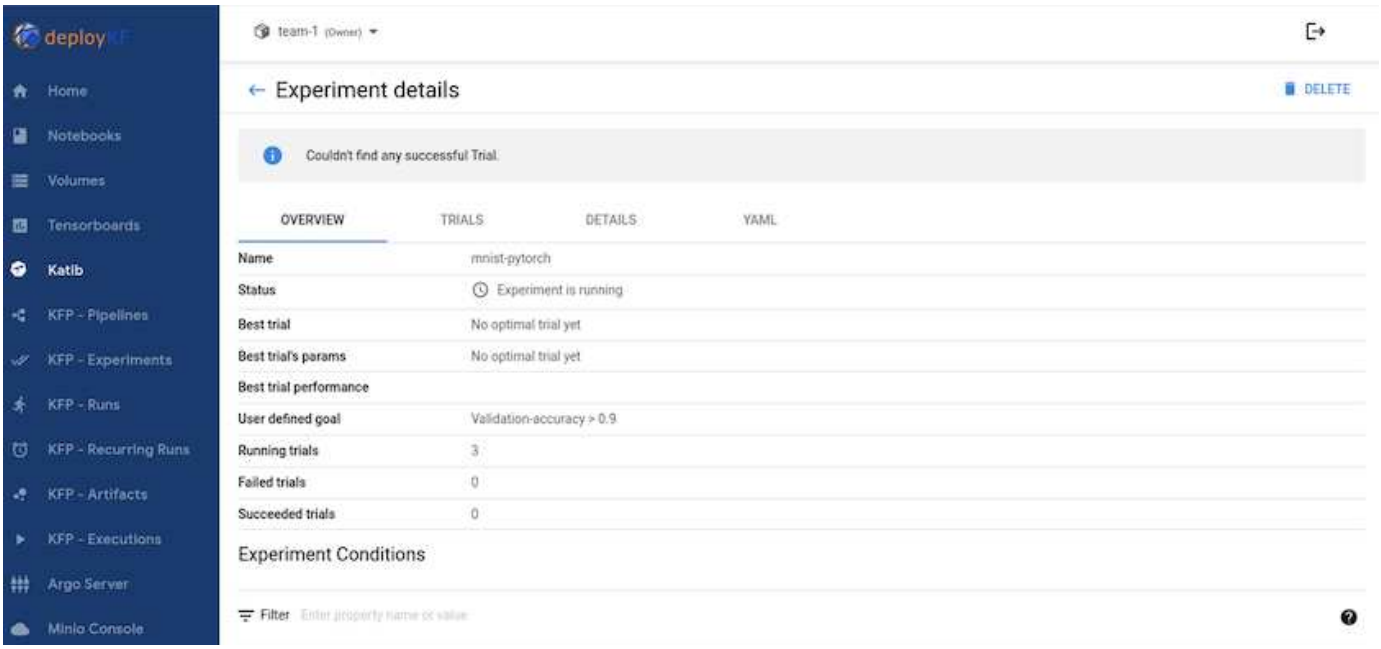
モデルのトレーニングが完了すると、Tensorboardを使用して結果を視覚化できます。 "Tensorboard" は、Kubeflowダッシュボードの機能として使用できます。ジョブ用のカスタムテンソルボードを作成できます。以下の例は、トレーニングの精度とエポック数、トレーニング損失とエポックの数。



Katibを使用したハイパーパラメータの実験

"カティブ" は、Kubeflow内のツールで、モデルのハイパーパラメータを試すために使用できます。実験を作成するには、まず目的のメトリック/ゴールを定義します。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、再生するハイパーパラメータを選択します（optimizer/learning_rate/number of layers）。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。または、必要な仕様で*YAML*ファイルを定義することもできます。以下はKatib実験の図です。





NetAppスナップショットを使用してデータを保存し、トレーサビリティを確保

モデルのトレーニング中に、トレーサビリティ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したい場合があります。これを行うには、以下の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向け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)
```

を参照してください "KubeflowのNetApp DataOpsツールキットの例" を参照してください。

Apache の通気

Apache Airflow の導入

このセクションでは、 Kubernetes クラスタ内に通気を導入するために完了しておく必要のあるタスクについて説明します。

Kubernetes 以外のプラットフォームに通気を導入することも可能です。Kubernetes 以外のプラットフォームに通気を導入することは、この解決策の範囲外です。

前提条件

このセクションで説明する導入の演習を行う前に、次の作業をすでに実行していることを前提としています。

1. Kubernetes クラスタをすでに使用している。
2. KubernetesクラスタにNetApp Astra Tridentをインストールして設定しておきます。Astra Tridentの詳細については、 "Astra Trident のドキュメント"。

Helm をインストールします

エアフローは、 Kubernetes の一般的なパッケージマネージャである Helm を使用して導入されます。エアーフローを導入する前に、導入ジャンプホストに Helm をインストールする必要があります。Helm を配置ジャンプホストにインストールするには、に従ってください "インストール手順" Helm の公式ドキュメントを参照してください。

デフォルトの **Kubernetes StorageClass** を設定します

通気を導入する前に、 Kubernetes クラスタ内にデフォルトのストレージクラスを指定する必要があります。エアフロー導入プロセスでは、デフォルトのストレージクラス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ます。StorageClass がデフォルトの StorageClass として指定されていない場合、導入は失敗します。クラスタ内でデフォルトのストレージクラスを指定するには、 "Kubeflow の導入" セクション。クラスタ内ですでにデフォルトの StorageClass を指定している場合は、この手順を省略できます。

Helm を使用してエアフローを展開します

Helm を使用して Kubernetes クラスタに通気を導入するには、導入ジャンプホストから次のタスクを実行します。

1. Helm を使用してエアフローを導入します。に従ってください "導入手順" アーティファクトハブの公式エアフロー図については、を参照してください。以下のコマンド例は、 Helm を使用したエアーフローの配置を示しています。必要に応じて 'custom-values/yaml ファイルの値を変更 ' 追加 ' または削除します

```
$ 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
```

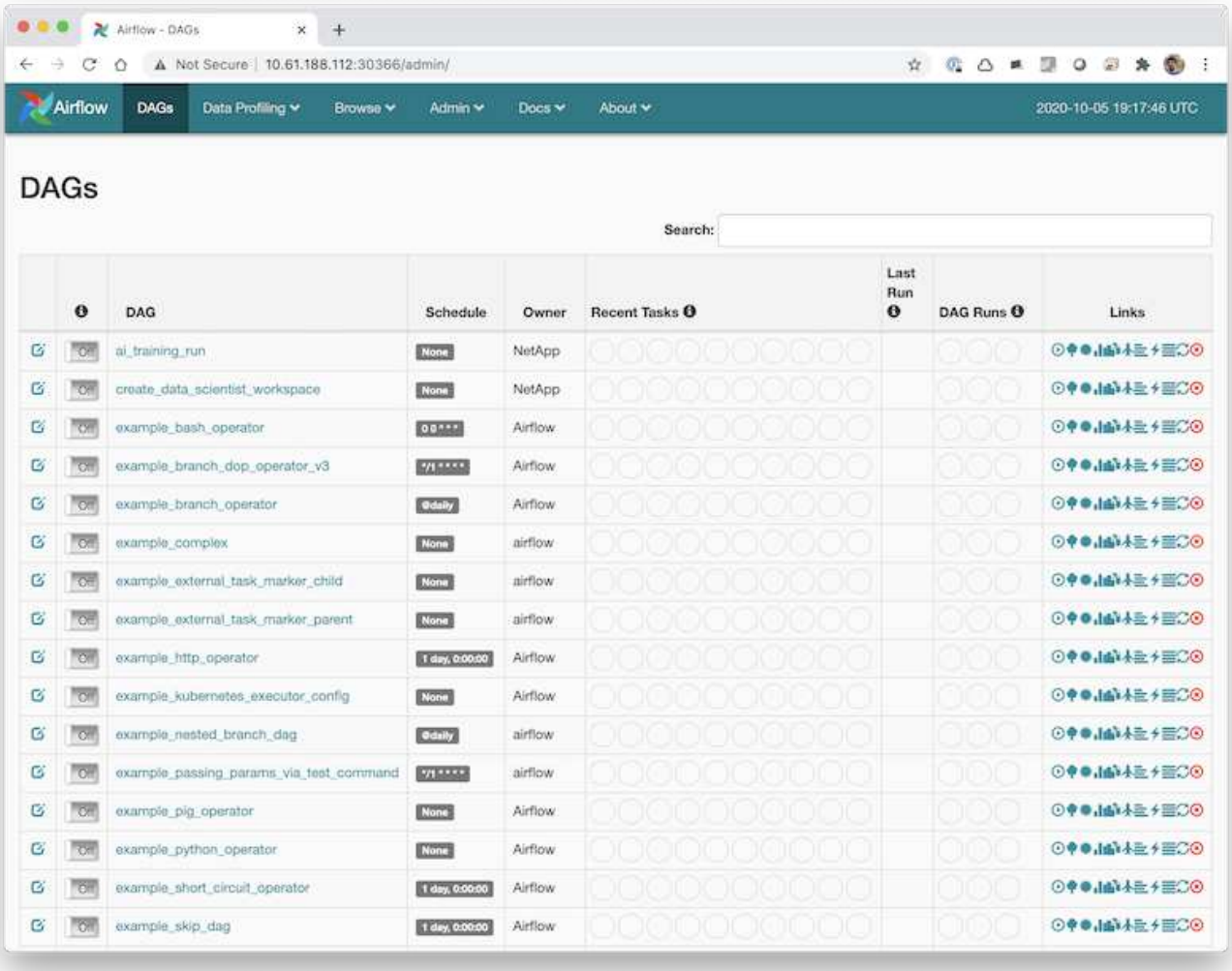
2. すべての通気ポッドが稼働中であることを確認します。すべてのポッドが起動するまでに数分かかる場合があります。

```
$ 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 の Helm を使用してエアーフローを導入したときにコンソールに出力された指示に従って、エアーフロー Web サービスの 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. 通気 Web サービスにアクセスできることを確認します。



エアーフローでの**NetApp DataOps Toolkit**の使用

。 "Kubernetes向けNetApp DataOpsツールキット" エアーフローと組み合わせて使用できます。NetApp DataOps ToolkitとAirflowを使用すると、Snapshotやクローンの作成などのNetAppのデータ管理処理を、Airflowによってオーケストレーションされた自動ワークフローに組み込むことができます。

を参照してください "通気の例" 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。

Astra Tridentの処理例

このセクションでは、Astra Tridentで実行するさまざまな処理の例を紹介します。

既存のボリュームをインポートします

ネットアップストレージシステム / プラットフォーム上に Kubernetes クラスタ内のコンテナにマウントする必要のある既存のボリュームがあり、クラスタ内の PVC に関連付けられていない場合は、それらのボリュー

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は、 Trident のボリュームインポート機能を使用してインポートできます。

次のコマンド例は、という名前のボリュームのインポートを示しています。 `pb_fg_all`。PVC の詳細については、を参照してください "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、を参照してください "Trident のドキュメント"。

「 accessModes 」の値「 ReadOnlyMany 」は、 PVC 仕様ファイルの例で指定されています。「 accessMode 」フィールドの詳細については、を参照してください "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します

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

kubectlを使用した新しいボリュームのプロビジョニング

次のコマンド例は、kubectlを使用した新しいFlexVolボリュームのプロビジョニングを示しています。

次の PVC 定義ファイル例では 'accessModes' の値 ReadWriteMany が指定されています「 accessMode 」フィールドの詳細については、を参照してください "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 Toolkitを使用した新しいボリュームのプロビジョニング

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では、Tridentを使用してボリュームをプロビジョニングしますが、ユーザのプロセスは簡易化されます。を参照してください "ドキュメント" を参照してください。

AIPod導入時のハイパフォーマンスジョブの例

シングルノードの AI ワークロードを実行

Kubernetes クラスタでシングルノードの AI ジョブと ML ジョブを実行するには、導入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、数ペタバイトのデータが含まれる可能性のあるデータボリュームをすばやく簡単に作成し、 Kubernetes のワークロードからアクセスできます。Kubernetes ポッド内からこのようなデータボリュームにアクセスできるようにするには、ポッドの定義で PVC を指定します。

このセクションでは、 Kubernetes クラスタで実行しようとしている特定の AI および ML ワークロードを（ Docker コンテナ形式で）コンテナ化済みであ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 ジョブを作成する方法を示しています。ImageNet データセットの詳細については、を参照してください "ImageNet の Web サイト"。

このジョブ例では、 8 個の GPU を要求するため、 8 個以上の GPU を搭載した 1 つの GPU ワーカーノードで実行することができます。このジョブ例は、 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を使用しているクラスタで送信できます。その場合、そのようなワーカーノードが使用可能になるまで、ジョブは保留状態のままになります。

また、ストレージ帯域幅を最大限にするために、必要なトレーニングデータを含むボリュームが、このジョブで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マウントされています。この 2 つ目のボリュームには、結果と指標を格納します。これらのボリュームは、 PVC の名前を使用してジョブ定義内で参照されます。Kubernetes ジョブの詳細については、を参照してください "Kubernetes の公式ドキュメント"。

「 M emory 」の値が「 emory 」である「 emptyDir 」ボリュームは、この例のジョブで作成されるポッド内の「 /dev/shm 」にマウントされます。Docker コンテナランタイムによって自動的に作成される「 /dev/shm 」仮想ボリュームのデフォルトサイズが、 TensorFlow のニーズに十分でない場合があります。次の例のように 'emptyDir' ボリュームをマウントすると '/dev/shm' 仮想ボリュームが十分に大きくなります「 emptyDir 」ボリュームの詳細については、を参照してください "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 'ecurityContext> 特権値 'true' が与えられますこの値は、コンテナにホスト上のルートアクセス権があることを意味します。このアノテーションは、実行中の特定のワークロードにルートアクセスが必要なために使用されます。具体的には、ワークロードで実行されるクリアキャッシュ処理にはルートアクセスが必要です。これが特権 : true の注釈であるかどうかは ' 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。次のコマンド例では、ジョブ定義で指定したとおりにジョブ用にポッドが 1 つ作成され、このポッドが GPU ワーカーノードの 1 つで現在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 1 で作成したジョブオブジェクトの削除を示しています。

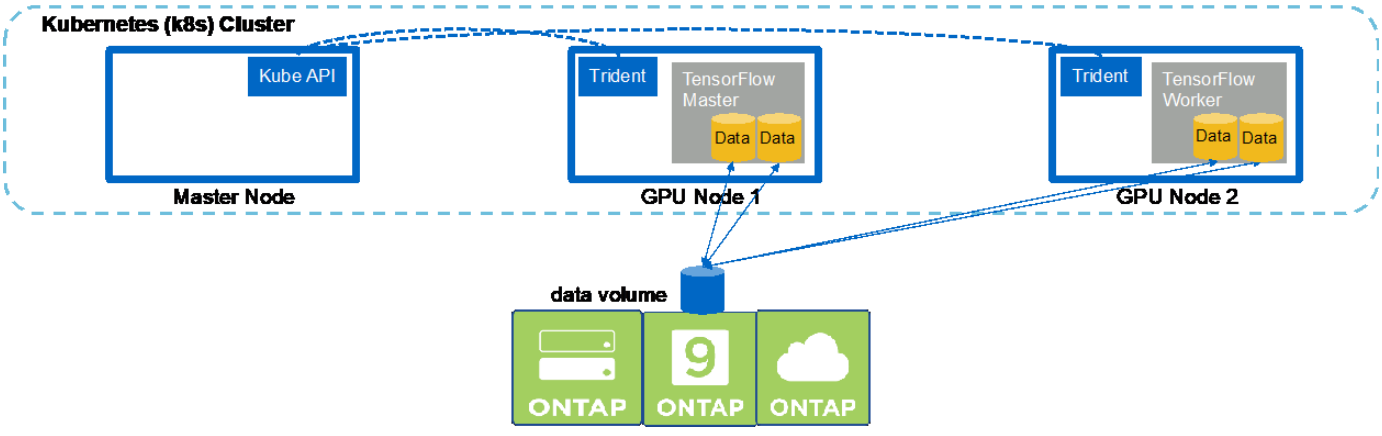
   ジョブオブジェクトを削除すると、関連付けられているポッドは Kubernetes によって自動的に削除されます。

```
$ 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.
```

分散型 AI の同期ワークロードを実行します

Kubernetes クラスタでマルチノードの AI と ML の同期ジョブを実行するには、導入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、ネットアップボリュームに格納されているデータを活用し、 1 つのワーカーノードで提供されるものよりも多くの GPU を使用することができます。同期分散 AI ジョブの説明については、次の図を参照してください。

同期分散ジョブを使用すると、非同期分散ジョブに比べてパフォーマンスとトレーニングの精度が向上します。同期ジョブと非同期ジョブの長所と短所については、本ドキュメントでは説明していません。



1. 次のコマンド例は、 1 つのワーカーの作成を示しています これは、同じの同期分散実行に関与します 1 つのノードで実行された TensorFlow ベンチマークジョブ を参照してください "シングルノードの AI ワークロードを実行"。この例では、 2 つのワーカーノードでジョブが実行されるため、導入されるワーカーは 1 つだけです。

この例のワーカー導入では、 8 個の GPU を要求し、 8 個以上の GPU を搭載した 1 つの GPU ワーカーノードで実行できます。GPU ワーカーノードが 8 個以上の GPU を搭載している場合、パフォーマンスを最大化するには、この数をワーカーノードが機能する GPU の数と同じにすると便利です。Kubernetes の導入の詳細については、を参照してください "Kubernetes の公式ドキュメント"。

この例では、このコンテナ化された特定のワーカーが自分で完了することはないため、 Kubernetes 環境が作成されます。そのため、 Kubernetes のジョブ構造を使用して導入することは理にかなっていません。従業員が自分で設計または作成した場合、作業構成を使用して従業員を配置することが理にかなっている場合があります。

この配置例の仕様で指定されているポッドには 'hostNetwork' の値が true に設定されていますこの値は、ポッドが、 Kubernetes が各ポッドに通常作成する仮想ネットワーキングスタックではなく、ホストワーカーノードのネットワークスタックを使用することを意味します。このアノテーションは、特定のワークロードが Open MPI 、 NCCL 、 Horovod を使用して同期分散方法でワークロードを実行するために使用されます。そのため、ホストネットワークスタックにアクセスする必要があります。Open MPI 、 NCCL 、および Horovod についての説明は、本ドキュメントの範囲外です。この hostNetwork:true' 注釈が必要かどうかは ' 実行している特定のワークロードの要件によって決まります「 hostNetwork 」フィールドの詳細については、を参照してください "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導入が正常に起動したことを確認します。次のコマンド例は、導入定義に示すように、単一のワーカーポッドが導入用に作成されたこと、およびこのポッドが GPU ワーカーノードの 1 つで現在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. マルチノード同期ジョブの実行を開始して参加させ、追跡するマスター用の Kubernetes ジョブを作成します。次のコマンド例では、 1 つのマスターを作成します。このマスターは、セクションの例で 1 つのノード上で実行された、同じ TensorFlow ベンチマークジョブの同期分散実行を追跡し、開始します "シングルノードの AI ワークロードを実行"。

この例では、マスタージョブは 8 個の GPU を要求するため、 8 個以上の GPU を搭載した 1 つの GPU ワーカーノードで実行できます。GPU ワーカーノードが 8 個以上の GPU を搭載している場合、パフォーマンスを最大化するには、この数をワーカーノードが機能する GPU の数と同じにすると便利です。

この例のジョブ定義で指定されているマスターポッドには、手順 1 でワーカーポッドに「 hostNetwork 」の値「 true 」が与えられたのと同様に、「 hostNetwork 」の値が「 true 」に設定されます。この値が必要な理由については、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。次のコマンド例では、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 GPU ワーカーノードの 1 つで現在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中で、マスターポッドとワーカーポッドが別々の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. 不要になったワーカー配置を削除します。次のコマンド例は、手順 1 で作成したワーカー配置オブジェクトの削除を示しています。

   ワーカー導入オブジェクトを削除すると、関連付けられているワーカーポッドは Kubernetes によって自動的に削除されます。

```
$ 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 で作成し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、関連付けられているマスターポッドは Kubernetes によって自動的に削除されます。

```
$ 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

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用権を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用権については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、http://www.netapp.com/TMに記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。