오픈 소스 **MLOps** 및 **NetApp**

NetApp Solutions

NetApp
May 10, 2024

목차

오픈 소스 **MLOps** 및 **NetApp**

오픈 소스 **MLOps** 및 **NetApp**

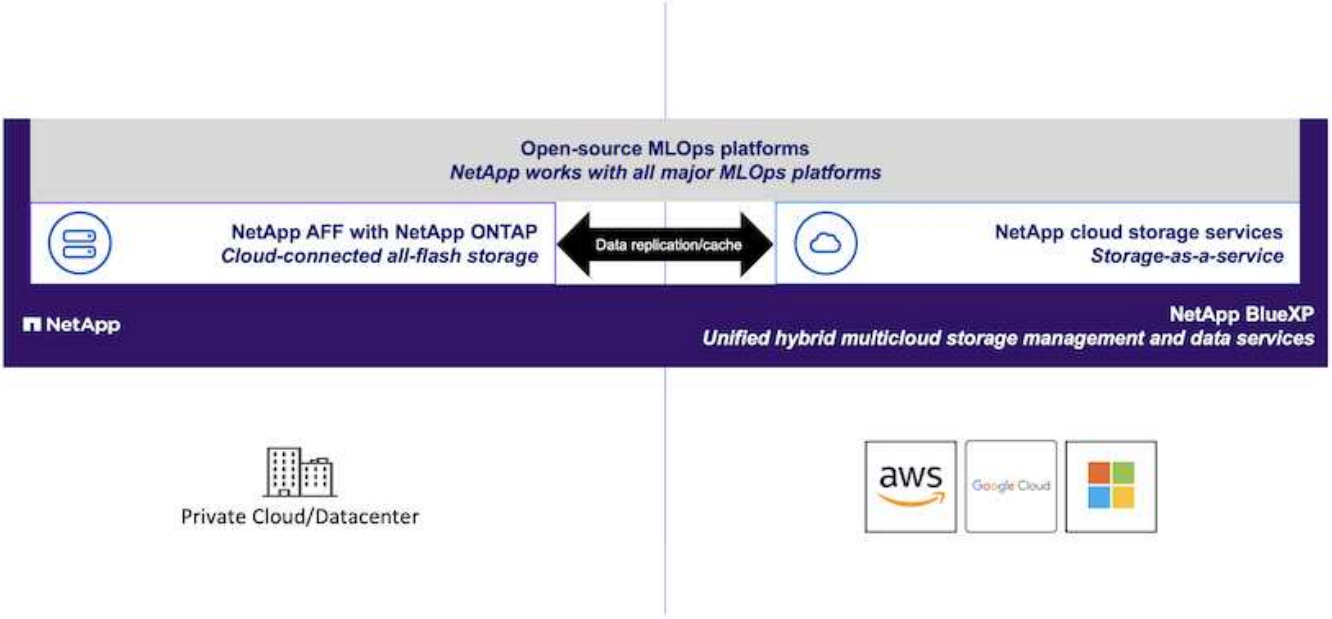
Mike Oglesby, NetApp에서 직접 지원합니다
모한 아차리아, NetApp

모든 규모와 업종에 상관없이 모든 기업과 조직은 실제 문제를 해결하고 혁신적인 제품과 서비스를 제공하며 경쟁이 갈수록 치열해지는 시장에서 경쟁 우위를 확보하기 위해 인공 지능(AI), 머신 러닝(ML), 딥 러닝(DL)으로 눈을 돌리고 있습니다. AI, ML 및 DL의 사용이 증가함에 따라 워크로드 확장성 및 데이터 가용성을 비롯한 많은 과제에 직면하게 됩니다. 이 솔루션은 NetApp 데이터 관리 기능을 주요 오픈 소스 툴 및 프레임워크와 결합하여 이러한 과제를 해결하는 방법을 보여줍니다.

이 솔루션은 MLOps 워크플로에 통합할 수 있는 여러 가지 오픈 소스 툴 및 프레임워크를 시연하기 위한 것입니다. 이러한 서로 다른 툴과 프레임워크는 요구사항 및 사용 사례에 따라 함께 사용하거나 단독으로 사용할 수 있습니다.

이 솔루션에서 다루는 툴/프레임워크는 다음과 같습니다.

- "아파치 기류"
- "Kubeflow"



기술 개요

인공 지능

AI는 컴퓨터가 인간의 마음의 인지 기능을 모방하도록 훈련되는 컴퓨터 과학 분야입니다. AI 개발자는 컴퓨터를 교육하여 사람과 유사하거나 훨씬 뛰어난 방식으로 문제를 배우고 해결합니다. 딥 러닝 및 머신 러닝은 AI의 하위 필드입니다. 조직은 중요한 비즈니스 요구사항을 지원하기 위해 AI, ML 및 DL을 점점 더 채택하고 있습니다. 몇 가지

예는 다음과 같습니다.

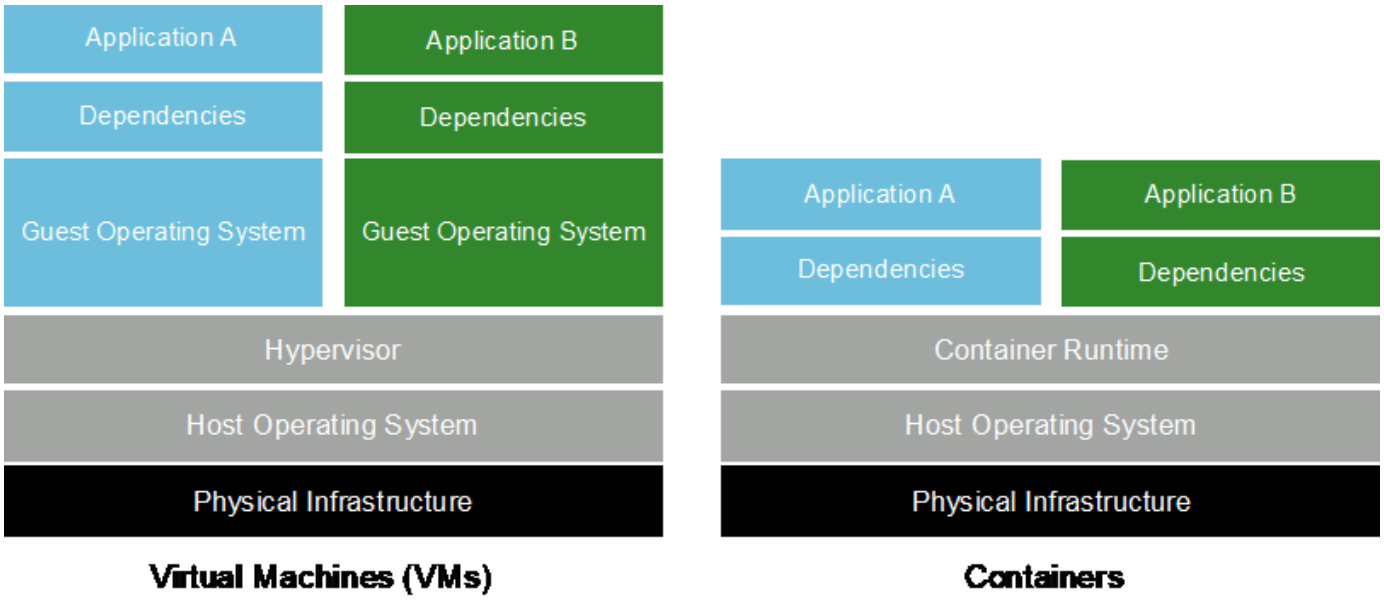
- 방대한 양의 데이터를 분석하여 이전에 알려지지 않은 비즈니스 인사이트를 도출합니다
- 자연어 처리를 사용하여 고객과 직접 상호 작용
- 다양한 비즈니스 프로세스 및 기능 자동화

최신 AI 훈련 및 추론 워크로드에는 대규모 병렬 컴퓨팅 기능이 필요합니다. 따라서 GPU의 병렬 처리 기능이 범용 CPU보다 훨씬 뛰어나기 때문에 AI 작업을 실행하는 데 GPU가 점점 더 많이 사용되고 있습니다.

컨테이너

컨테이너는 공유 호스트 운영 체제 커널 위에서 실행되는 격리된 사용자 공간 인스턴스입니다. 컨테이너 채택이 빠르게 증가하고 있습니다. 컨테이너는 가상 머신(VM)이 제공하는 것과 동일한 애플리케이션 샌드박스(sandbox)의 많은 이점을 제공합니다. 하지만 VM이 사용하는 하이퍼바이저 및 게스트 운영 체제 계층이 없어졌기 때문에 컨테이너는 훨씬 더 가볍습니다. 다음 그림에서는 가상 시스템과 컨테이너를 보여 줍니다.

또한 컨테이너를 사용하면 애플리케이션 종속성, 실행 시간 등을 애플리케이션과 직접 효율적으로 패키징할 수 있습니다. 가장 일반적으로 사용되는 컨테이너 패키징 형식은 Docker 컨테이너입니다. Docker 컨테이너 형식으로 컨테이너화된 애플리케이션은 Docker 컨테이너를 실행할 수 있는 모든 시스템에서 실행할 수 있습니다. 모든 종속성이 컨테이너 자체에 패키지되어 있기 때문에 응용 프로그램의 종속성이 컴퓨터에 없는 경우에도 마찬가지입니다. 자세한 내용은 를 참조하십시오 "Docker 웹 사이트".



쿠버네티스

Kubernetes는 Google에서 원래 설계한 개방형 소스, 분산형 컨테이너 오케스트레이션 플랫폼으로, 현재 CNCF(Cloud Native Computing Foundation)에서 관리하고 있습니다. Kubernetes는 컨테이너화된 애플리케이션의 구축, 관리, 확장 기능을 자동화할 수 있습니다. 최근 몇 년 동안 Kubernetes는 주요 컨테이너 오케스트레이션 플랫폼으로 부상했습니다. 자세한 내용은 를 참조하십시오 "Kubernetes 웹 사이트".

NetApp Astra Trident

Astra Trident를 사용하면 ONTAP(AFF, NetApp FAS, Select, 클라우드, NetApp ONTAP용 Amazon FSx), Element 소프트웨어(NetApp HCI, SolidFire), Azure NetApp Files 서비스 및 Google Cloud 기반 Cloud Volumes Service

Astra Trident는 Kubernetes와 기본적으로 통합되는 CSI(컨테이너 스토리지 인터페이스) 호환 동적 스토리지 오케스트레이터입니다.

NetApp DataOps 툴킷

를 클릭합니다 "NetApp DataOps 툴킷" Python 기반 툴로서, 고성능 스케일아웃 NetApp 스토리지에서 지원하는 개발/훈련 작업 공간과 추론 서버의 관리를 간소화합니다. 주요 기능은 다음과 같습니다.

- 고성능 스케일아웃 NetApp 스토리지의 지원을 받는 새로운 고용량 작업 공간을 빠르게 프로비저닝합니다.
- 실험이나 신속한 반복을 위해 고용량 작업 공간을 거의 동시에 복제합니다.
- 백업 및/또는 추적/기준선 설정을 위해 대용량 작업 공간의 스냅샷을 거의 동시에 저장합니다.
- 대용량 고성능 데이터 볼륨을 거의 동시에 프로비저닝, 복제 및 스냅샷으로 제공합니다.

Kubeflow

Kubeflow는 Google에서 원래 개발한 Kubernetes용 오픈 소스 AI 및 ML 툴킷입니다. Kubeflow 프로젝트를 통해 Kubernetes에서 AI 및 ML 워크플로우를 간단하게 배포, 이식 및 확장할 수 있습니다. Kubeflow는 복잡한 Kubernetes를 추상화하여 데이터 과학자가 가장 잘 아는 — 데이터 과학에 집중할 수 있도록 지원합니다. 시각화는 다음 그림을 참조하십시오. Kubeflow는 올인원 MLOps 플랫폼을 선호하는 조직에 적합한 오픈 소스 옵션입니다. 자세한 내용은 를 참조하십시오 "Kubeflow 웹 사이트".

Kubeflow 파이프라인

Kubeflow 파이프라인은 Kubeflow의 핵심 구성 요소입니다. Kubeflow 파이프라인은 이식 가능하고 확장 가능한 AI 및 ML 워크플로우를 정의하고 배포하기 위한 플랫폼 및 표준입니다. 자세한 내용은 를 참조하십시오 "Kubeflow 공식 문서".

Jupyter 노트북 서버

Jupyter Notebook Server는 데이터 과학자가 실시간 코드와 설명이 포함된 Jupyter Notebooks라는 위키 형식의 문서를 만들 수 있는 오픈 소스 웹 애플리케이션입니다. Jupyter Notebooks는 AI 및 ML 프로젝트를 문서화, 저장, 공유하는 수단으로 AI 및 ML 커뮤니티에서 널리 사용되고 있습니다. Kubeflow는 Kubernetes에서 Jupyter Notebook Server의 프로비저닝 및 구축을 단순화합니다. Jupyter Notebooks에 대한 자세한 내용은 를 참조하십시오 "Jupyter 웹 사이트". Kubeflow와 관련하여 Jupyter Notebooks에 대한 자세한 내용은 를 참조하십시오 "Kubeflow 공식 문서".

케이티비주식회사

Katib은 자동 머신 러닝(AutoML)을 위한 Kubernetes 네이티브 프로젝트입니다. Katib는 하이퍼매개변수 조정, 조기 중지 및 신경 아키텍처 검색(NAS)을 지원합니다. Katib은 머신 러닝(ML) 프레임워크에 제약을 받지 않는 프로젝트입니다. 사용자가 선택한 모든 언어로 작성된 애플리케이션의 하이퍼 매개 변수를 조정할 수 있으며 TensorFlow, MXNet, PyTorch, XGBoost, 있습니다. Katib는 Bayesian 최적화, Parzen Estimators의 Tree of Parzen Estimators, Random Search, CoVariance Matrix 어댑테이션 진화 전략, Hyperband, 효율적 신경 아키텍처 검색, 차별화 아키텍처 검색 등과 같은 다양한 AutoML 알고리즘을 많이 지원합니다. Kubeflow와 관련하여 Jupyter Notebooks에 대한 자세한 내용은 를 참조하십시오 "Kubeflow 공식 문서".

아파치 기류

Apache Airflow는 복잡한 엔터프라이즈 워크플로우를 프로그래밍 방식으로 작성, 스케줄링 및 모니터링할 수 있는 오픈 소스 워크플로우 관리 플랫폼입니다. ETL 및 데이터 파이프라인 워크플로우를 자동화하는 데 주로 사용되지만, 이러한 유형의 워크플로우에만 국한되지 않습니다. Airbnb가 공기 흐름 프로젝트를 시작했지만 그 이후 업계에서 매우 인기를 끌며 현재는 Apache Software Foundation의 후원으로 자리 잡았습니다. Python으로 공기 흐름을 작성하고 Python

스크립트를 통해 공기 흐름을 생성하고 "코드로 구성"이라는 원칙에 따라 공기 흐름을 설계할 수 있습니다. 많은 엔터프라이즈 공기 흐름 사용자가 이제 Kubernetes에서 공기 흐름을 실행합니다.

유도된 **DAG(Acclic Graphs)**

공기 흐름에서 워크플로우는 DAG(Directed Acyclic Graphs)라고 합니다. DAG는 DAG 정의에 따라 순차적으로, 병렬로 또는 둘의 조합으로 실행되는 작업으로 구성됩니다. 공기 흐름 스케줄러는 DAG 정의에 지정된 작업 수준 종속성을 준수하여 일련의 작업자에 대해 개별 작업을 실행합니다. DAG는 Python 스크립트를 통해 정의 및 생성됩니다.

NetApp ONTAP를 참조하십시오

NetApp의 최신 세대 스토리지 관리 소프트웨어인 ONTAP 9는 기업이 인프라를 현대화하고 클라우드 지원 데이터 센터로 전환할 수 있도록 지원합니다. ONTAP는 업계 최고 수준의 데이터 관리 기능을 활용하여 데이터가 상주하는 위치와 상관없이 단일 툴셋으로 데이터를 관리하고 보호할 수 있습니다. 필요에 따라 에지, 코어, 클라우드 등 어느 위치로도 데이터를 자유롭게 이동할 수 있습니다. ONTAP 9에는 데이터 관리를 단순화하고, 중요 데이터를 더 빨리 처리하고, 보호하며, 하이브리드 클라우드 아키텍처 전체에서 차세대 인프라 기능을 지원하는 다양한 기능이 포함되어 있습니다.

데이터 관리를 단순화하십시오

데이터 관리는 AI 애플리케이션에 적합한 리소스를 사용하고 AI/ML 데이터 세트를 교육할 수 있도록 엔터프라이즈 IT 운영 및 데이터 과학자에게 매우 중요합니다. NetApp 기술에 대한 다음 추가 정보는 이 검증의 범위에 포함되지 않지만, 배포에 따라 달라질 수 있습니다.

ONTAP 데이터 관리 소프트웨어에는 운영을 간소화 및 단순화하고 총 운영 비용을 절감하는 다음과 같은 기능이 있습니다.

- 인라인 데이터 컴팩션 및 확대된 중복제거: 데이터 컴팩션은 스토리지 블록 내부의 낭비되는 공간을 줄이고, 중복제거는 실제 용량을 상당히 늘려줍니다. 이는 로컬에 저장된 데이터와 클라우드로 계층화된 데이터에 적용됩니다.
- 최소, 최대 및 적응형 서비스 품질(AQoS): 세부적인 서비스 품질(QoS) 제어로 고도의 공유 환경에서 중요 애플리케이션의 성능 수준을 유지할 수 있습니다.
- NetApp FabricPool를 참조하십시오. AWS(Amazon Web Services), Azure, NetApp StorageGRID 스토리지 솔루션을 포함한 퍼블릭 클라우드 및 프라이빗 클라우드 스토리지에 콜드 데이터를 자동으로 계층화합니다. FabricPool에 대한 자세한 내용은 를 참조하십시오 "TR-4598: FabricPool 모범 사례".

데이터 가속화 및 보호

ONTAP는 탁월한 수준의 성능과 데이터 보호를 제공하며 다음과 같은 방법으로 이러한 기능을 확장합니다.

- 성능 및 짧은 지연 시간: ONTAP는 가장 짧은 지연 시간으로 가장 높은 처리량을 제공합니다.
- 데이터 보호: ONTAP는 모든 플랫폼에서 공통 관리를 지원하는 내장 데이터 보호 기능을 제공합니다.
- NVE(NetApp 볼륨 암호화). ONTAP는 온보드 및 외부 키 관리를 모두 지원하는 기본 볼륨 레벨 암호화를 제공합니다.
- 멀티테넌시 및 다단계 인증. ONTAP를 사용하면 인프라 리소스를 최고 수준의 보안으로 공유할 수 있습니다.

미래 지향형 인프라

ONTAP은 다음과 같은 기능을 통해 끊임없이 변화하는 까다로운 비즈니스 요구사항을 충족할 수 있도록 지원합니다.

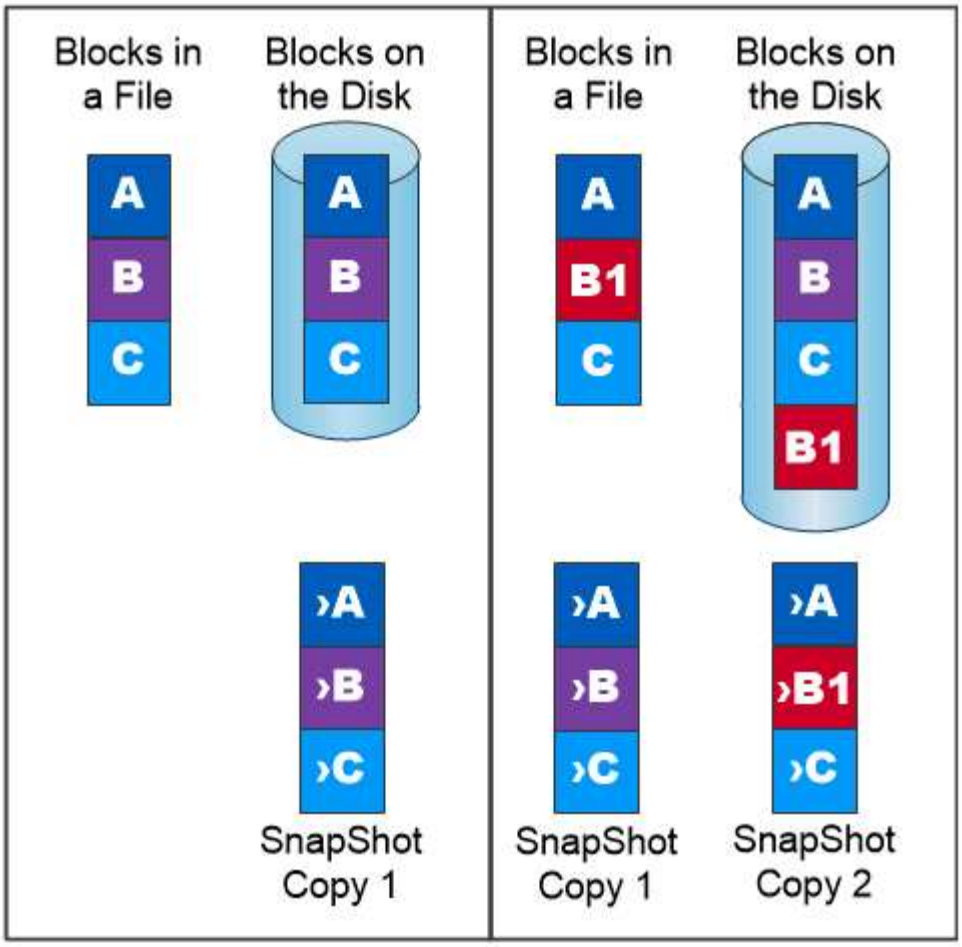
- 원활한 확장 및 무중단 운영: ONTAP은 운영 중단 없이 기존 컨트롤러 및 스케일아웃 클러스터에 용량을 추가할 수 있도록 지원합니다. 고객은 고비용이 따르는 데이터 마이그레이션이나 운영 중단 없이 NVMe 및 32Gb FC와 같은 최신 기술로 업그레이드할 수 있습니다.
- 클라우드 연결: ONTAP은 주요 클라우드와 연결되는 스토리지 관리 소프트웨어로, 모든 퍼블릭 클라우드에서 소프트웨어 정의 스토리지 및 클라우드 네이티브 인스턴스 옵션이 제공됩니다.
- 새로운 애플리케이션과 통합: ONTAP은 기존 엔터프라이즈 앱을 지원하는 인프라와 동일한 인프라를 사용하여 자율주행 차량, 스마트 시티, Industry 4.0과 같은 차세대 플랫폼 및 애플리케이션을 위한 엔터프라이즈급 데이터 서비스를 제공합니다.

NetApp Snapshot 복사본

NetApp 스냅샷 복사본은 볼륨의 읽기 전용 시점 이미지입니다. 다음 그림과 같이 이미지는 스토리지 공간을 최소한으로 사용하고, 마지막 스냅샷 복사본 생성 이후 생성된 파일의 변경사항만 기록하므로 경미한 성능 오버헤드를 발생시킵니다.

스냅샷 복사본은 핵심 ONTAP 스토리지 가상화 기술인 WAFL(Write Anywhere File Layout)의 효율성을 높여줍니다. 데이터베이스와 마찬가지로 WAFL는 메타데이터를 사용하여 디스크의 실제 데이터 블록을 가리킵니다. 하지만 WAFL은 데이터베이스와 달리 기존 블록을 덮어쓰지 않습니다. 업데이트된 데이터를 새 블록에 쓰고 메타데이터를 변경합니다. ONTAP은 데이터 블록을 복사하는 대신 스냅샷 복사본을 생성할 때 메타데이터를 참조하므로 스냅샷 복사본이 매우 효율적입니다. 이렇게 하면 복사할 블록을 찾는 데 다른 시스템이 발생하는 탐색 시간과 복사본 자체를 만드는 비용이 제거됩니다.

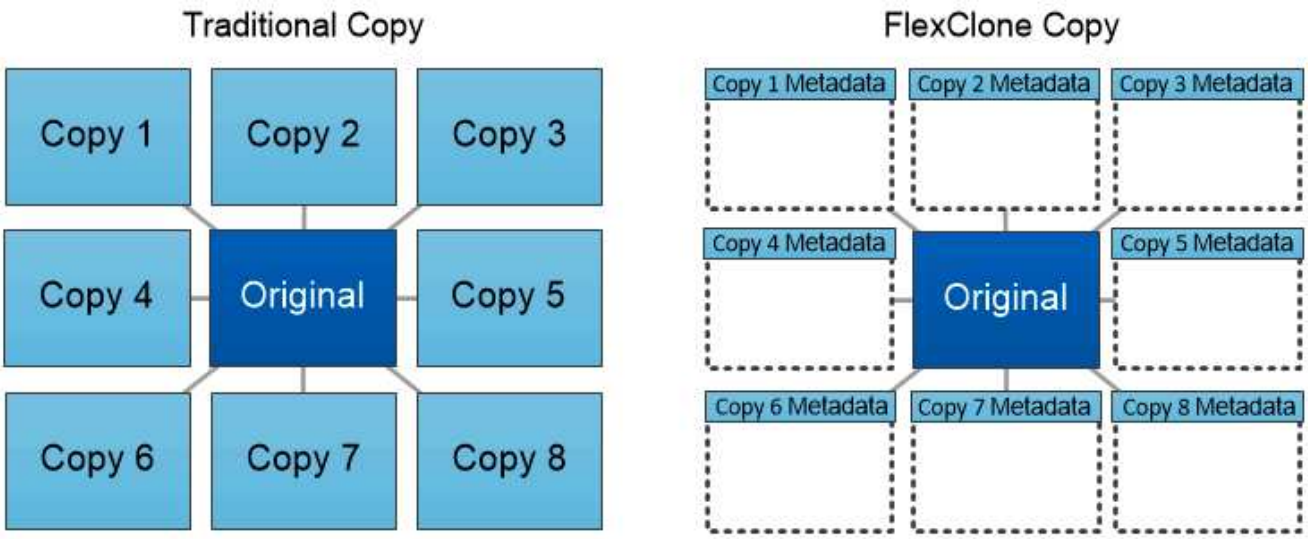
스냅샷 복사본을 사용하여 개별 파일 또는 LUN을 복구하거나 볼륨의 전체 콘텐츠를 복원할 수 있습니다. ONTAP은 스냅샷 복사본의 포인터 정보를 디스크의 데이터와 비교하여 다운타임 또는 상당한 성능 비용 없이 누락 또는 손상된 개체를 재구성합니다.



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

NetApp FlexClone 기술

NetApp FlexClone 기술은 Snapshot 메타데이터를 참조하여 볼륨의 쓰기 가능한 특정 시점 복사본을 생성합니다. 복사본은 다음 그림과 같이 복사본에 변경 사항이 기록될 때까지 메타데이터에 필요한 사항을 제외하고 데이터 블록을 부모와 공유하고 스토리지를 사용하지 않습니다. FlexClone 소프트웨어를 사용하면 기존 복사본을 생성하는 데 몇 분 또는 몇 시간이 걸릴 수 있으며 최대 규모의 데이터 세트도 거의 즉시 복사할 수 있습니다. 따라서 동일한 데이터 세트의 여러 복사본(예: 개발 작업 공간)이 필요하거나 데이터 세트의 임시 복사본(운영 데이터 세트에 대해 애플리케이션 테스트)이 필요한 경우에 적합합니다.



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

NetApp SnapMirror 데이터 복제 기술

NetApp SnapMirror 소프트웨어는 Data Fabric에서 사용하기 쉬운 비용 효율적인 통합 복제 솔루션입니다. LAN 또는 WAN을 통해 데이터를 고속으로 복제합니다. 가상 환경과 기존 환경 모두에서 비즈니스 크리티컬 애플리케이션을 포함한 모든 유형의 애플리케이션에 대해 높은 데이터 가용성과 빠른 데이터 복제를 제공합니다. 하나 이상의 NetApp 스토리지 시스템에 데이터를 복제하고 2차 데이터를 지속적으로 업데이트함으로써 데이터가 최신 상태로 유지되고 필요할 때마다 사용할 수 있으며 외부 복제 서버가 필요하지 않습니다. 다음 그림은 SnapMirror 기술을 활용하는 아키텍처의 예입니다.

SnapMirror 소프트웨어는 변경된 블록만 네트워크를 통해 전송함으로서 NetApp ONTAP 스토리지 효율성을 활용합니다. SnapMirror 소프트웨어는 또한 내장된 네트워크 압축 기능을 사용하여 데이터 전송을 더 신속하게 수행하고 네트워크 대역폭 활용률을 70%까지 줄입니다. SnapMirror 기술을 사용하면 하나의 씬 복제 데이터 스트림을 활용하여 활성 미러와 이전 시점의 복사본을 둘 다 유지 관리하는 단일 저장소를 만들 수 있으므로 네트워크 트래픽이 최대 50% 감소합니다.

NetApp BlueXP 복사 및 동기화

BlueXP 복사 및 동기화는 빠르고 안전한 데이터 동기화를 제공하는 NetApp 서비스입니다. 온프레미스 NFS 또는 SMB 파일 공유 간에 파일을 전송해야 하는 경우, NetApp StorageGRID, NetApp ONTAP S3, NetApp Cloud Volumes Service, Azure NetApp Files, AWS S3, AWS EFS, Azure Blob, Google Cloud Storage, 즉 IBM Cloud Object Storage, BlueXP Copy and Sync는 필요한 파일을 빠르고 안전하게 이동합니다.

데이터가 전송되면 소스와 타겟 모두에서 사용할 수 있습니다. BlueXP 복사 및 동기화는 업데이트가 트리거되거나 미리 정의된 일정에 따라 데이터가 지속적으로 동기화되는 경우 필요 시 데이터를 동기화할 수 있습니다. BlueXP Copy 및 Sync는 변경된 부분만 이동하므로 데이터 복제에 소비되는 시간과 비용이 최소화됩니다.

BlueXP Copy and Sync는 매우 간단하게 설정하고 사용할 수 있는 서비스형 소프트웨어(SaaS) 툴입니다. BlueXP Copy 및 Sync에 의해 트리거되는 데이터 전송은 데이터 브로커에 의해 수행됩니다. BlueXP Copy 및 Sync 데이터 브로커는 AWS, Azure, Google Cloud Platform 또는 사내에 구축할 수 있습니다.

NetApp XCP

NetApp XCP는 모든 NetApp 및 NetApp 간 데이터 마이그레이션 및 파일 시스템 통찰력을 위한 클라이언트 기반 소프트웨어입니다. xCP는 사용 가능한 모든 시스템 리소스를 활용하여 대용량 데이터 세트 및 고성능 마이그레이션을 처리함으로써 최대한의 성능을 발휘하도록 설계되었습니다. xCP를 사용하면 보고서를 생성하는 옵션을 통해 파일 시스템에 대한 완벽한 가시성을 확보할 수 있습니다.

NetApp XCP는 NFS 및 SMB 프로토콜을 지원하는 단일 패키지로 제공됩니다. xCP에는 NFS 데이터 세트용 Linux 바이너리와 SMB 데이터 세트용 Windows 실행 파일이 포함되어 있습니다.

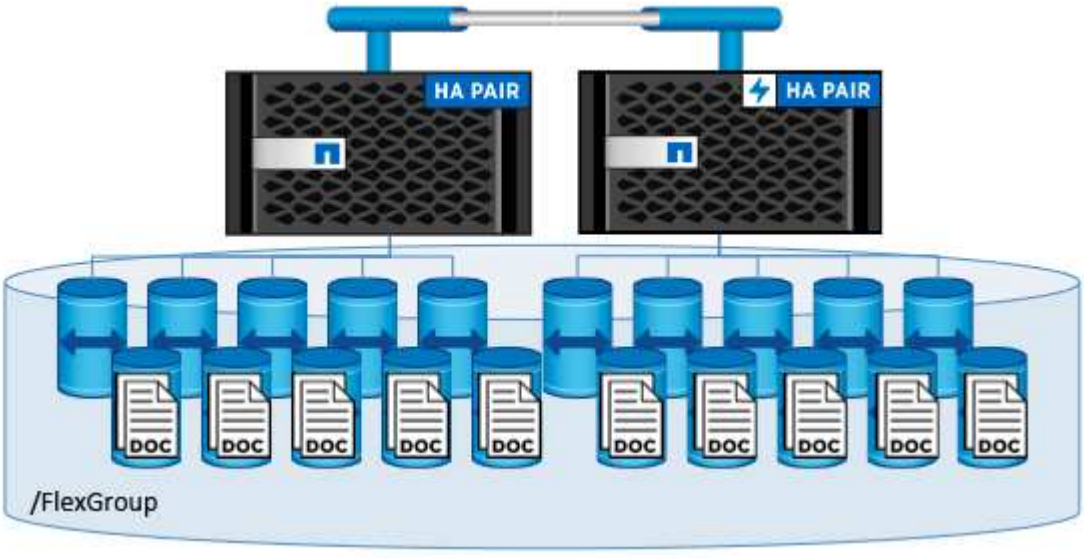
NetApp XCP File Analytics는 파일 공유를 감지하고 파일 시스템에서 스캔을 실행하며 파일 분석을 위한 대시보드를 제공하는 호스트 기반 소프트웨어입니다. XCP File Analytics는 NetApp 및 타사 시스템과 모두 호환되며 Linux 또는 Windows 호스트에서 실행되어 NFS 및 SMB에서 내보낸 파일 시스템에 대한 분석 기능을 제공합니다.

NetApp ONTAP FlexGroup 볼륨

교육 데이터 세트는 잠재적으로 수십억 개의 파일로 구성됩니다. 파일에는 텍스트, 오디오, 비디오 및 기타 형식의 비정형 데이터가 포함될 수 있으며, 이 데이터를 병렬로 읽고 저장해야 합니다. 스토리지 시스템은 수많은 작은 파일을 저장해야 하며 순차적 I/O 및 랜덤 I/O를 위해 병렬로 이들 파일을 읽어야 합니다

FlexGroup 볼륨은 다음 그림과 같이 여러 개의 구성 멤버 볼륨으로 구성된 단일 네임스페이스입니다. 스토리지 관리자 관점에서 FlexGroup 볼륨은 NetApp FlexVol 볼륨과 마찬가지로 관리되고 작동합니다. FlexGroup 볼륨의 파일은 개별 구성원 볼륨에 할당되며 볼륨 또는 노드에 스트라이핑되지 않습니다. 다음과 같은 기능을 지원합니다.

- FlexGroup 볼륨은 메타데이터가 많은 워크로드에 수 페타바이트에 달하는 용량과 예측 가능한 짧은 지연 시간을 제공합니다.
- 동일한 네임스페이스에서 최대 4천억 개의 파일을 지원합니다.
- CPU, 노드, 애그리게이트, 구성 FlexVol 볼륨에서 NAS 워크로드에 병렬 작업을 지원합니다.



있습니다

이 솔루션은 특정 하드웨어에 종속되지 않습니다. 이 솔루션은 Trident에서 지원하는 모든

NetApp 물리적 스토리지 어플라이언스, 소프트웨어 정의 인스턴스 또는 클라우드 서비스와 호환됩니다. NetApp AFF 스토리지 시스템, Amazon FSx for NetApp ONTAP, Azure NetApp Files 또는 NetApp Cloud Volumes ONTAP 인스턴스가 여기에 포함됩니다. 또한 사용되는 Kubernetes 버전이 Kubeflow 및 NetApp Astra Trident에서 지원되는 경우 모든 Kubernetes 클러스터에서 솔루션을 구현할 수 있습니다. Kubeflow에서 지원하는 Kubernetes 버전 목록은 를 참조하십시오 "Kubeflow 공식 문서". Trident에서 지원하는 Kubernetes 버전 목록은 를 참조하십시오 "Trident 문서". 솔루션을 검증하는 데 사용된 환경에 대한 자세한 내용은 다음 표를 참조하십시오.

소프트웨어 구성 요소	버전
아파치 기류	2.0.1
Apache Airflow Helm Chart(Apache Airflow 제어 차트	8.0.8
Kubeflow	1.7, 를 통해 배포 "구축KF" 0.1.1
쿠버네티스	1.26
NetApp Astra Trident	2007년 3월

지원

NetApp은 Apache Airflow, Kubeflow 또는 Kubernetes에 대한 엔터프라이즈 지원을 제공하지 않습니다. 완전히 지원되는 MLOps 플랫폼에 관심이 있다면 "NetApp에 문의하십시오" NetApp이 파트너와 공동으로 제공하는 모든 MLOps 솔루션에 대해 설명합니다.

NetApp Astra Trident 구성

NetApp AIPod 구축을 위한 Astra Trident 백엔드의 예

Astra Trident를 사용하여 Kubernetes 클러스터 내에서 스토리지 리소스를 동적으로 프로비저닝하려면 먼저 하나 이상의 Trident 백엔드를 생성해야 합니다. 다음 예제는 에서 이 솔루션의 구성 요소를 배포할 때 만들 수 있는 여러 가지 유형의 백엔드를 나타냅니다 "NetApp AIPod". 백엔드에 대한 자세한 내용은 을 참조하십시오 "Astra Trident 문서".

1. NetApp은 AIPod에 대해 FlexGroup 지원 Trident 백엔드를 생성할 것을 권장합니다.

   다음 명령은 AIPod 스토리지 가상 머신(SVM)에 대한 FlexGroup 지원 Trident 백엔드의 생성을 보여줍니다. 이 백엔드는 를 사용합니다 `ontap-nas-flexgroup` 스토리지 드라이버. ONTAP는 FlexVol와 FlexGroup의 두 가지 기본 데이터 볼륨 유형을 지원합니다. FlexVol 볼륨의 크기는 제한되어 있습니다(이 쓰기 작업 시 최대 크기는 특정 구축에 따라 다름). 반면 FlexGroup 볼륨은 최대 20PB 및 4천억 개 파일까지 선형적으로 확장할 수 있으므로 데이터 관리를 크게 간소화하는 단일 네임스페이스를 제공합니다. 따라서 FlexGroup 볼륨은 대량의 데이터를 사용하는 AI 및 ML 워크로드에 최적화되어 있습니다.

   소량의 데이터로 작업하고 FlexGroup 볼륨 대신 FlexVol 볼륨을 사용하려는 경우, ONTAP-NAS-Flexgroup 스토리지 드라이버 대신 'ONTAP-NAS' 스토리지 드라이버를 사용하는 Trident 백엔드를 생성할 수 있습니다.

```
$ 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. 또한 NetApp은 FlexVol 지원 Trident 백엔드를 생성할 것을 권장합니다. 영구 응용 프로그램 호스팅, 결과 저장, 출력, 디버그 정보 등을 위해 FlexVol 볼륨을 사용할 수 있습니다. FlexVol 볼륨을 사용하려면 하나 이상의 FlexVol 지원 Trident 백엔드를 생성해야 합니다. 다음 명령의 예는 단일 FlexVol 지원 Trident 백엔드의 생성을 보여줍니다.

```
$ 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 구축을 위한 Kubernetes StorageClasses 예

Astra Trident를 사용하여 Kubernetes 클러스터 내에서 스토리지 리소스를 동적으로 프로비저닝하려면 먼저 하나 이상의 Kubernetes StorageClasses를 생성해야 합니다. 다음 예제는 에서 이 솔루션의 구성 요소를 배포할 때 만들 수 있는 다양한 유형의 StorageClasses를 나타냅니다 "NetApp AIPod". StorageClasses에 대한 자세한 내용은 를 참조하십시오 "Astra

Trident 문서".

1. NetApp은 섹션에서 생성한 FlexGroup 지원 Trident 백엔드에 대한 StorageClass를 생성할 것을 권장합니다 "NetApp AIPod 구축을 위한 Astra Trident 백엔드의 예", 1단계. 다음 예제 명령은 섹션에서 만든 두 예제 백엔드에 해당하는 여러 StorageClasses를 만드는 방법을 보여 줍니다 "NetApp AIPod 구축을 위한 Astra Trident 백엔드의 예", 단계 1 - 를 활용하는 단계 "RDMA 상의 NFS" 그리고 그렇지 않은 것.

   영구 볼륨은 해당 PersistentVolumeClaim(PVC)이 삭제되어도 삭제되지 않도록 다음 예에서는 "Retain"의 "reclaimPolicy" 값을 사용합니다. '청구 정책' 필드에 대한 자세한 내용은 공식 을 참조하십시오 "Kubernetes 문서".

   참고: 다음 예제 StorageClasses는 최대 전송 크기인 262144를 사용합니다. 이 최대 전송 크기를 사용하려면 그에 따라 ONTAP 시스템에서 최대 전송 크기를 구성해야 합니다. 을 참조하십시오 "ONTAP 설명서" 를 참조하십시오.

   참고: RDMA를 통해 NFS를 사용하려면 ONTAP 시스템에서 NFS over RDMA를 구성해야 합니다. 자세한 내용은 ONTAP documentation 링크를 참조하십시오.

   참고: 다음 예제에서 StorageClass 정의 파일의 StoragePool 필드에 특정 백엔드가 지정되지 않았습니다.

```
$ 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. 또한 섹션에서 생성한 FlexVol 지원 Trident 백엔드에 해당하는 StorageClass를 생성하는 것이 좋습니다 "AIPod 구축을 위한 Astra Trident 백엔드의 예", 2단계. 다음 명령 예에서는 FlexVol 볼륨에 대한 단일 StorageClass를 생성하는 것을 보여 줍니다.

   참고: 다음 예제에서 StorageClass 정의 파일의 StoragePool 필드에 특정 백엔드가 지정되지 않았습니다. Kubernetes를 사용하여 이 StorageClass를 사용하여 볼륨을 관리하는 경우 Trident는 를 사용하는 사용 가능한 백엔드를 사용합니다 `ontap-nas` 드라이버.

```
$ 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
```

Kubeflow

Kubeflow 구축

이 섹션에서는 Kubernetes 클러스터에 Kubeflow를 구축하기 위해 완료해야 하는 작업에 대해 설명합니다.

필수 구성 요소

이 섹션에 요약된 배포 연습을 수행하기 전에 이미 다음 작업을 수행했다고 가정합니다.

1. 이미 작동 중인 Kubernetes 클러스터가 있으며, 구축하려는 Kubeflow 버전에서 지원하는 Kubernetes 버전을 실행하고 있습니다. 지원되는 Kubernetes 버전 목록은 에서 Kubeflow 버전에 대한 종속성을 참조하십시오 "Kubeflow 공식 문서".
2. Kubernetes 클러스터에 NetApp Astra Trident를 이미 설치하고 구성했습니다. Astra Trident에 대한 자세한 내용은 를 참조하십시오 "Astra Trident 문서".

기본 **Kubernetes StorageClass**를 설정합니다

Kubeflow를 구축하기 전에 Kubernetes 클러스터 내에 기본 StorageClass를 지정하는 것이 좋습니다. Kubeflow 구축 프로세스에서 기본 StorageClass를 사용하여 새 영구 볼륨 프로비저닝을 시도할 수 있습니다. 기본 StorageClass 로 지정된 StorageClass 가 없으면 배포가 실패할 수 있습니다. 클러스터 내에서 기본 StorageClass를 지정하려면 배포 점프 호스트에서 다음 작업을 수행합니다. 클러스터 내에서 기본 StorageClass를 이미 지정한 경우에는 이 단계를 건너뛸 수 있습니다.

1. 기존 StorageClasses 중 하나를 기본 StorageClass로 지정합니다. 다음 명령 예에서는 이름이 인 StorageClass 를 지정했습니다 `ontap-ai-flexvols-retain` 기본 StorageClass로 사용됩니다.

ONTAP-NAS-Flexgroup Trident 백엔드 유형은 PVC 크기가 매우 큽니다. 기본적으로 Kubeflow는 크기가 몇 GB인 PVC를 프로비저닝하려고 시도합니다. 따라서 Kubeflow 구축을 위해 "ONTAP-NAS-flexgroup" 백엔드 유형을 기본 StorageClass로 사용하는 StorageClass를 지정할 수 없습니다.

```
$ 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 구축 옵션

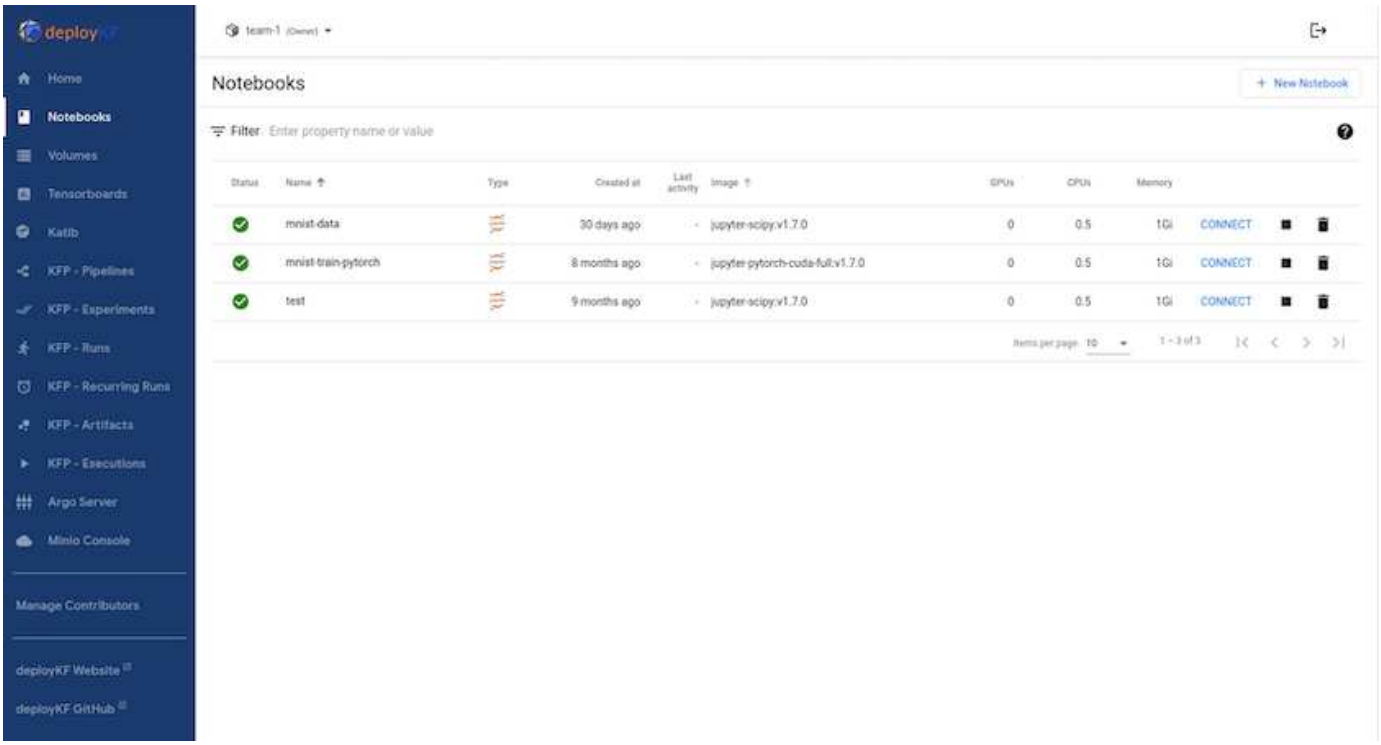
Kubeflow를 배포하는 다양한 옵션이 있습니다. 을 참조하십시오 "Kubeflow 공식 문서" 구축 옵션 목록을 확인하고 요구사항에 가장 적합한 옵션을 선택하십시오.

검증을 위해 를 사용하여 Kubeflow 1.7을 배포했습니다 "구축KF" 0.1.1.

Kubeflow 작업 및 작업 예

데이터 과학자 또는 개발자 사용을 위한 **Jupyter Notebook Workspace**를 제공합니다

Kubeflow는 새로운 Jupyter Notebook 서버를 신속하게 프로비저닝하여 데이터 과학자 작업 공간 역할을 할 수 있습니다. Kubeflow 컨텍스트 내의 Jupyter Notebooks에 대한 자세한 내용은 를 참조하십시오 "Kubeflow 공식 문서".



Kubeflow와 함께 **NetApp DataOps** 툴킷 사용

를 클릭합니다 "Kubernetes용 NetApp 데이터 과학 툴킷" Kubeflow와 함께 사용할 수 있습니다. Kubeflow와 함께 NetApp Data Science Toolkit을 사용하면 다음과 같은 이점이 있습니다.

- 데이터 과학자는 Jupyter Notebook 내에서 직접 스냅샷 및 클론 생성과 같은 고급 NetApp 데이터 관리 작업을 수행할 수 있습니다.
- 스냅샷 및 클론 생성과 같은 고급 NetApp 데이터 관리 작업은 Kubeflow 파이프라인 프레임워크를 사용하여 자동화된 워크플로에 통합할 수 있습니다.

을 참조하십시오 "Kubeflow 예" Kubeflow 기반 툴킷 사용에 대한 자세한 내용은 NetApp Data Science Toolkit GitHub 리포지토리 를 참조하십시오.

워크플로우 예 **- Kubeflow** 및 **NetApp DataOps** 툴킷을 사용하여 이미지 인식 모델을 학습합니다

이 섹션에서는 Kubeflow 및 NetApp DataOps 툴킷을 사용하여 이미지 인식을 위한 신경망 교육 및 배포에 관련된 단계를 설명합니다. 이 슬라이드의 목적은 NetApp 스토리지를 통합하는 교육 작업을 보여주는 예입니다.

필수 구성 요소

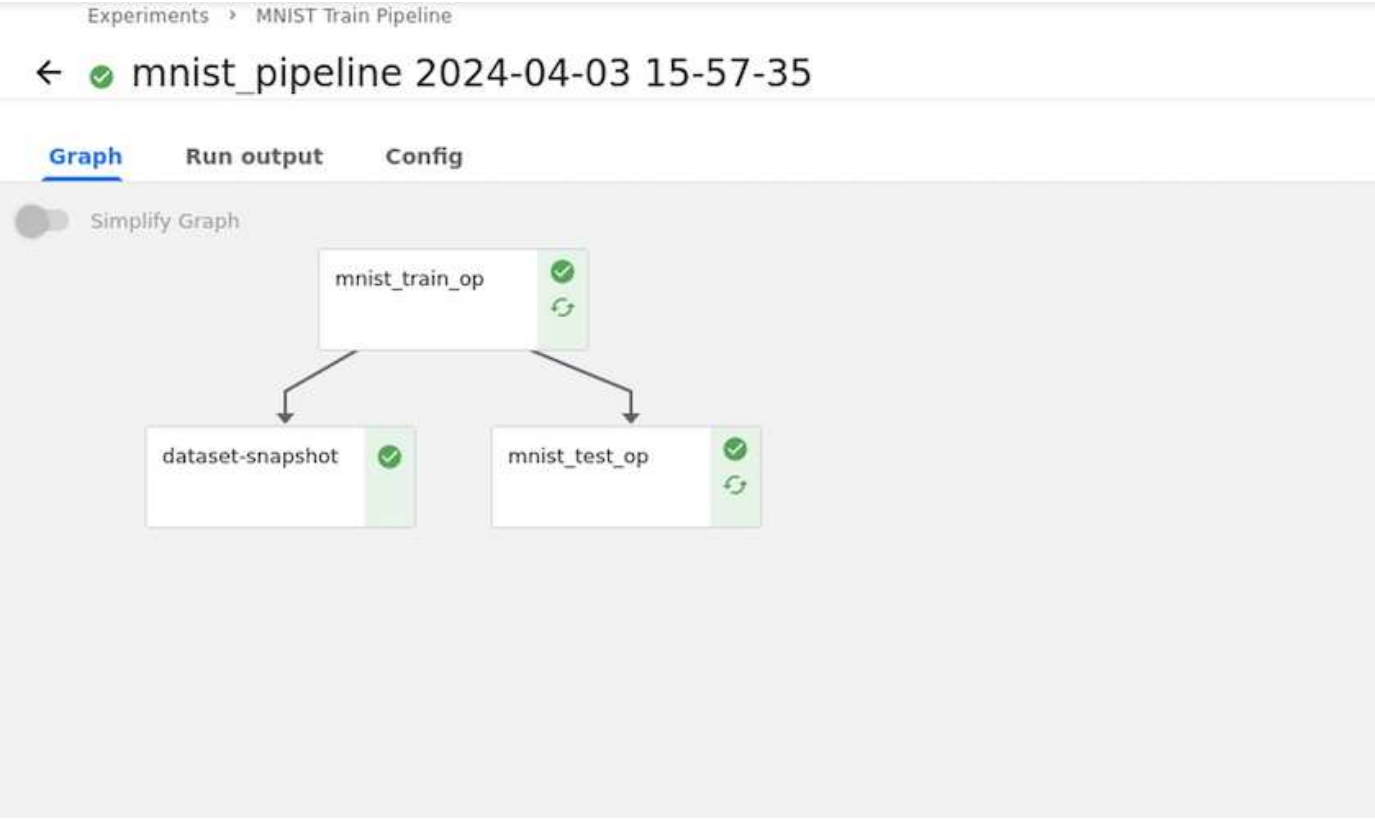
Kubeflow 파이프라인 내에서 기차 및 테스트 단계에 사용할 필수 구성으로 Dockerfile을 생성합니다.
다음은 Dockerfile의 예입니다.

```
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"]
```

요구 사항에 따라 프로그램을 실행하는 데 필요한 모든 필수 라이브러리와 패키지를 설치합니다. 기계 학습 모델을 교육하기 전에 이미 작동하는 Kubeflow 배포가 있다고 가정합니다.

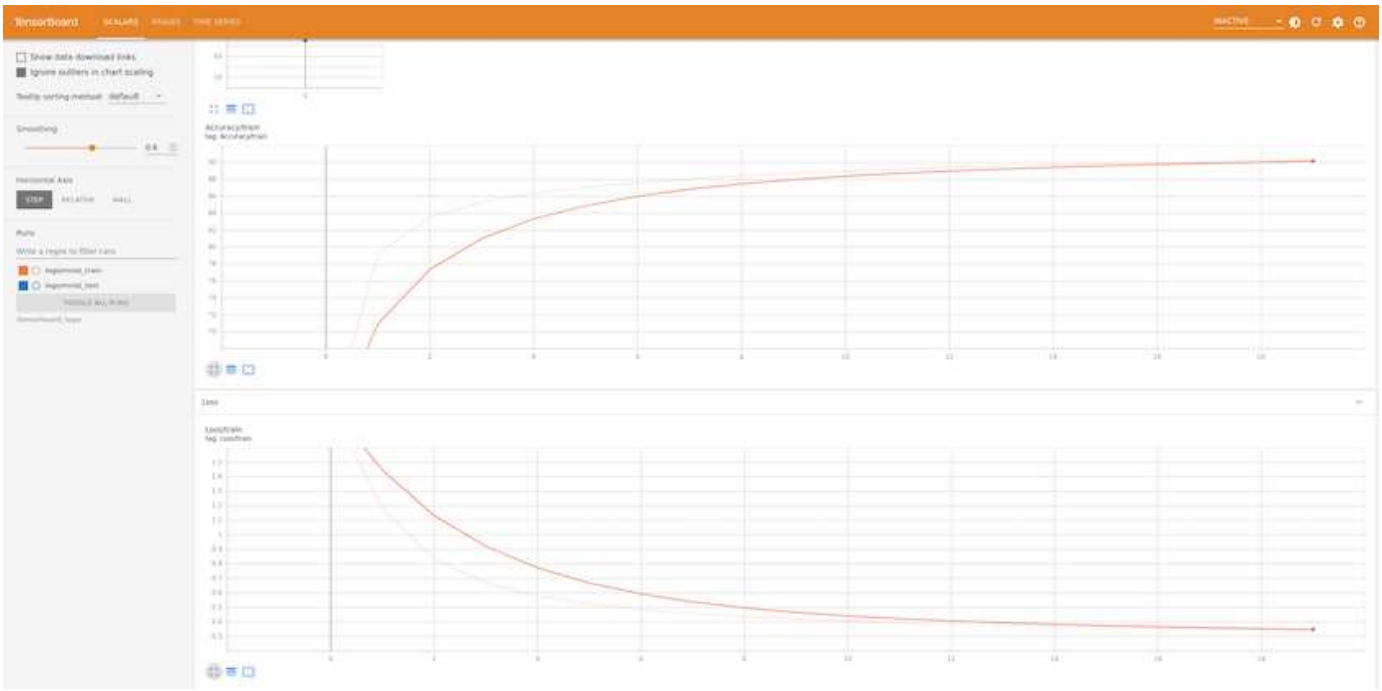
PyTorch 및 **Kubeflow** 파이프라인을 사용하여 **MNIST** 데이터에 대한 소규모 **NN**을 훈련하십시오

MNIST 데이터에 대해 훈련된 소규모 Neural Network의 예를 사용합니다. MNIST 데이터 세트는 0-9의 숫자 자필 이미지로 구성됩니다. 이미지 크기는 28x28픽셀입니다. 데이터 세트는 60,000개의 기차 이미지와 10,000개의 검증 이미지로 구분됩니다. 이 실험에 사용되는 신경망은 2계층 피드포워드 네트워크입니다. 교육은 Kubeflow 파이프라인을 사용하여 실행됩니다. 설명서를 참조하십시오 "여기" 를 참조하십시오. Kubeflow 파이프라인은 사전 요구 사항 섹션의 Docker 이미지를 통합합니다.



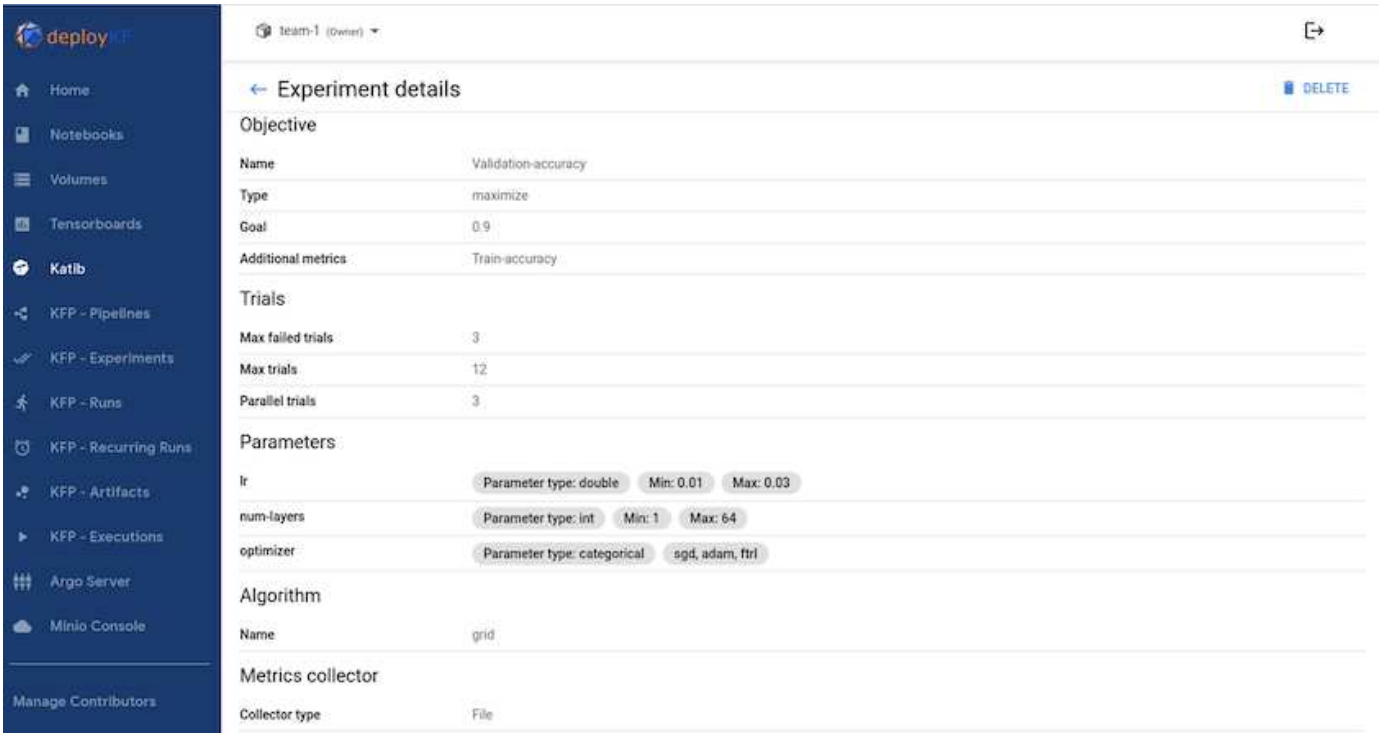
Tensorboard를 사용하여 결과를 시각화합니다

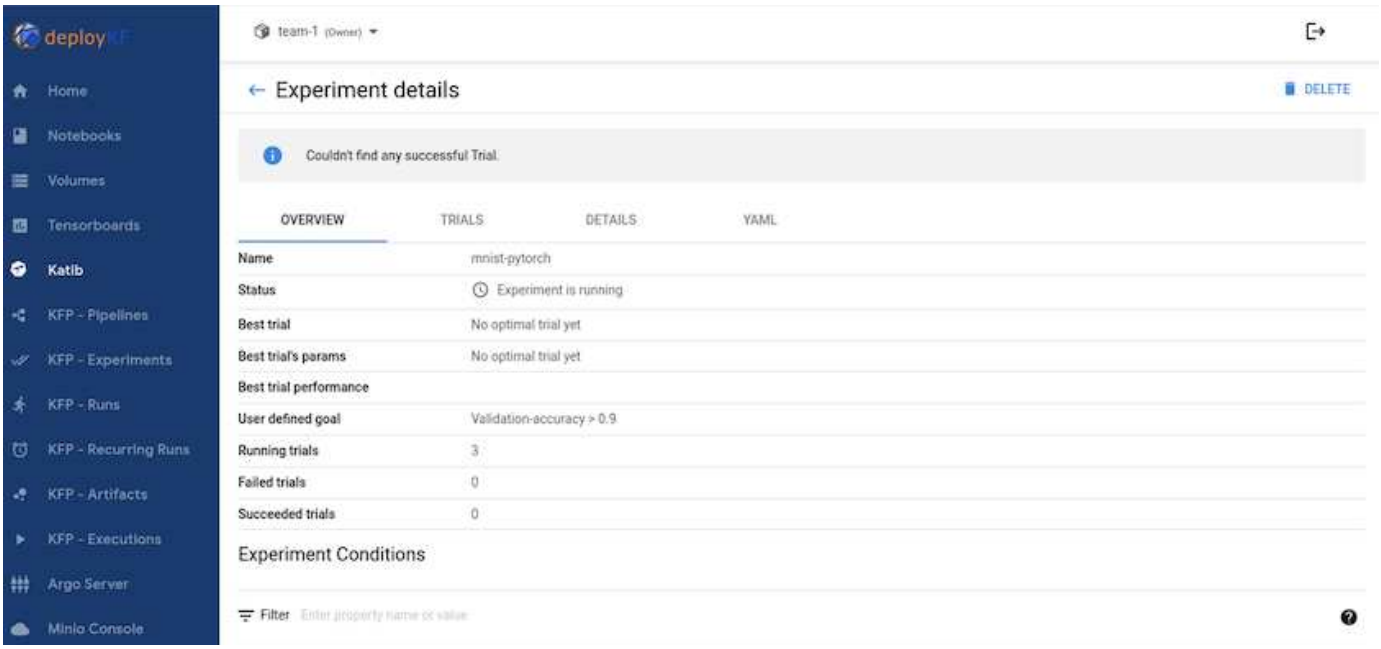
모델이 훈련되면 Tensorboard를 사용하여 결과를 시각화할 수 있습니다. "Tensorboard를 사용합니다" 은 Kubeflow Dashboard에서 기능으로 제공됩니다. 작업에 대한 사용자 정의 텐서보드를 작성할 수 있습니다. 아래 예는 과(와) 비교 트레이닝 정확도의 플롯을 보여줍니다 기 수 및 교육 손실 대 Epoch 수.



Katib를 사용하여 **Hyperparameters**를 실험합니다

"케이티비주식회사" 은 Kubeflow 내의 도구로서 모델 hyperparameters를 실험하는 데 사용할 수 있습니다. 실험을 생성하려면 먼저 원하는 메트릭/목표를 정의합니다. 이것은 일반적으로 테스트 정확도입니다. 메트릭이 정의되면 재생할 하이퍼파라미터(optimizer/learning_rate/layer 수)를 선택합니다. Katib는 사용자 정의 값으로 hyperparameter sweep을 수행하여 원하는 메트릭을 만족하는 최적의 매개 변수 조합을 찾습니다. UI의 각 섹션에서 이러한 매개 변수를 정의할 수 있습니다. 또는 필요한 사양으로 * YAML * 파일을 정의할 수도 있습니다. 다음은 Katib 실험의 그림입니다.





NetApp 스냅샷을 사용하여 추적을 위해 데이터를 저장합니다

모델 훈련 중에 추적 기능을 위해 훈련 데이터 세트의 스냅샷을 저장하고자 할 수 있습니다. 이를 위해 아래와 같이 파이프라인에 스냅샷 단계를 추가할 수 있습니다. 스냅샷을 생성하기 위해 를 사용할 수 있습니다 "Kubernetes용 NetApp DataOps 툴킷".

```
@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)
```

을 참조하십시오 "Kubeflow에 대한 NetApp DataOps 툴킷의 예" 를 참조하십시오.

아파치 기류

Apache Airflow Deployment

이 섹션에서는 Kubernetes 클러스터에 공기 흐름을 구축하기 위해 완료해야 하는 작업에 대해 설명합니다.

Kubernetes 이외의 플랫폼에 공기 흐름을 배포할 수 있습니다. Kubernetes가 아닌 다른 플랫폼에 공기 흐름을 배포하는 것은 이 솔루션의 범위를 벗어납니다.

필수 구성 요소

이 섹션에 요약된 배포 연습을 수행하기 전에 이미 다음 작업을 수행했다고 가정합니다.

1. Kubernetes 클러스터 작업이 이미 진행 중입니다.
2. Kubernetes 클러스터에 NetApp Astra Trident를 이미 설치하고 구성했습니다. Astra Trident에 대한 자세한 내용은 를 참조하십시오 "Astra Trident 문서".

제어 장치를 설치합니다

Kubernetes의 유명 패키지 매니저인 Helm을 사용하여 공기 흐름을 구축합니다. 공기 흐름을 배치하기 전에 배포 점프 호스트에 Helm을 설치해야 합니다. 배포 점프 호스트에 Helm을 설치하려면 에 따르십시오 "설치 지침" 공식 Helm 문서.

기본 **Kubernetes StorageClass**를 설정합니다

공기 흐름을 구축하기 전에 Kubernetes 클러스터 내에서 기본 StorageClass를 지정해야 합니다. Airflow 배포 프로세스는 기본 StorageClass를 사용하여 새 영구 볼륨의 프로비저닝을 시도합니다. 기본 StorageClass로 지정된 StorageClass가 없으면 배포가 실패합니다. 클러스터 내에서 기본 StorageClass를 지정하려면 에 설명된 지침을 따릅니다 "Kubeflow 구축" 섹션을 참조하십시오. 클러스터 내에서 기본 StorageClass를 이미 지정한 경우에는 이 단계를 건너뛸 수 있습니다.

Helm을 사용하여 공기 흐름을 전개하십시오

Helm을 사용하여 Kubernetes 클러스터에 공기 흐름을 배포하려면 배포 점프 호스트에서 다음 작업을 수행하십시오.

1. 에 따라 헬름으로 공기 흐름을 전개하십시오 "배포 지침" Artifact Hub의 공식 공기 흐름 도표입니다. 다음 명령 예는 Helm을 사용한 공기 흐름의 배치를 보여줍니다. 환경과 원하는 구성에 따라 필요에 따라 'custom-values.yAML' 파일에서 값을 수정, 추가 및/또는 제거합니다.

```
$ 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
```

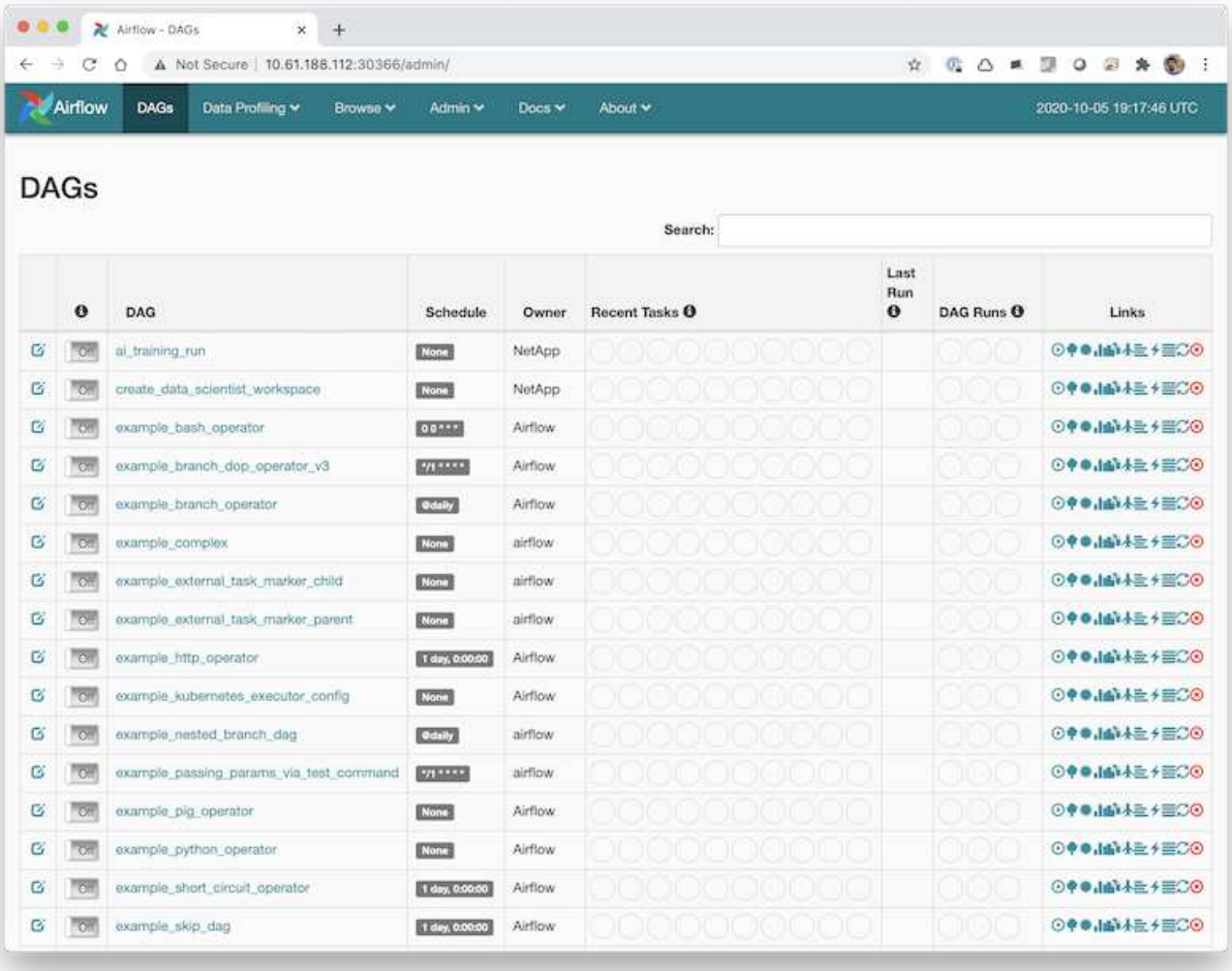
2. 모든 공기 흐름 포드가 실행 중인지 확인합니다. 모든 Pod를 시작하는 데 몇 분 정도 걸릴 수 있습니다.

```
$ 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. 1단계에서 제어 장치를 사용하여 공기 흐름을 배포할 때 콘솔에 인쇄된 지침에 따라 공기 흐름 웹 서비스 URL을 얻습니다.

```
$ 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 웹 서비스에 액세스할 수 있는지 확인합니다.



NetApp DataOps 툴킷을 공기 흐름과 함께 사용하십시오

를 클릭합니다 "Kubernetes용 NetApp DataOps 툴킷" 공기 흐름과 함께 사용할 수 있습니다. NetApp DataOps 툴킷을 공기 흐름과 함께 사용하면 스냅샷 및 클론 생성과 같은 NetApp 데이터 관리 작업을 공기 흐름으로 오케스트레이션되는 자동 워크플로우에 통합할 수 있습니다.

을 참조하십시오 "공기 흐름의 예" 섹션에서 NetApp DataOps 툴킷 사용에 대한 자세한 내용을 확인할 수 있습니다.

Astra Trident 운영의 예

이 섹션에는 Astra Trident에서 수행할 수 있는 다양한 작업의 예가 포함되어 있습니다.

기존 볼륨을 가져옵니다

Kubernetes 클러스터 내의 컨테이너에 마운트할 NetApp 스토리지 시스템/플랫폼에 기존 볼륨이 있지만, 클러스터의 PVC와 연결되지 않은 경우 이러한 볼륨을 가져와야 합니다. Trident 볼륨 가져오기 기능을 사용하여 이러한 볼륨을 가져올 수 있습니다.

다음 명령 예에서는 라는 이름의 볼륨을 가져오는 방법을 보여 줍니다 `pb_fg_all`. PVC에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서". 볼륨 가져오기 기능에 대한 자세한 내용은 를 참조하십시오 "Trident 문서".

예제 PVC 규격 파일에는 'ReadOnlyMany'의 'accessModes' 값이 지정되어 있습니다. 'accessMode' 필드에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서".

```
$ 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
```

새 볼륨을 프로비저닝합니다

Trident를 사용하여 NetApp 스토리지 시스템 또는 플랫폼에서 새 볼륨을 프로비저닝할 수 있습니다.

kubectl을 사용하여 새 볼륨을 프로비저닝합니다

다음 명령 예는 kubectl을 사용한 새 FlexVol 볼륨의 프로비저닝을 보여 줍니다.

다음 PVC 정의 파일에는 ReadWriteMany 의 accessModes 값이 지정되어 있습니다. 'accessMode' 필드에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서".

```
$ 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 툴킷을 사용하여 새 볼륨을 프로비저닝합니다

또한 Kubernetes용 NetApp DataOps 툴킷을 사용하여 NetApp 스토리지 시스템 또는 플랫폼에서 새 볼륨을 프로비저닝할 수 있습니다. Kubernetes용 NetApp DataOps 툴킷은 Trident를 활용하여 볼륨을 프로비저닝하지만 사용자 프로세스를 간소화합니다. 을 참조하십시오 "문서화" 를 참조하십시오.

AIPod 구축을 위한 고성능 작업의 예

단일 노드 AI 워크로드 실행

Kubernetes 클러스터에서 단일 노드 AI 및 ML 작업을 실행하려면 배포 점프 호스트에서 다음 작업을 수행하십시오. Trident를 사용하면 페타바이트에 이를 수 있는 데이터 볼륨을 빠르고 쉽게 만들어 Kubernetes 워크로드에 액세스할 수 있습니다. Kubernetes Pod에서 데이터 볼륨에 액세스할 수 있도록 하려면 POD 정의에 PVC를 지정하기만 하면 됩니다.

ⓘ 이 섹션에서는 Kubernetes 클러스터에서 실행하려고 하는 특정 AI 및 ML 워크로드를 이미 컨테이너화(Docker 컨테이너 형식)했다고 가정합니다.

1. 다음 명령 예는 ImageNet 데이터 세트를 사용하는 TensorFlow 벤치마크 워크로드에 대한 Kubernetes 작업 생성을 보여줍니다. ImageNet 데이터 세트에 대한 자세한 내용은 를 참조하십시오 "ImageNet 웹 사이트".

   이 예시 작업은 8개의 GPU를 요청하므로 8개 이상의 GPU를 갖춘 단일 GPU 작업자 노드에서 실행할 수 있습니다. 이 예시 작업은 8개 이상의 GPU를 갖춘 작업자 노드가 없거나 현재 다른 워크로드를 사용 중인 클러스터에 제출할 수 있습니다. 이 경우 해당 작업자 노드를 사용할 수 있을 때까지 작업은 보류 중 상태로 유지됩니다.

   또한 스토리지 대역폭을 최대화하기 위해 필요한 교육 데이터가 들어 있는 볼륨이 이 작업에서 생성되는 POD 내에 두 번 마운트됩니다. 포드에도 다른 볼륨이 마운트됩니다. 이 두 번째 볼륨은 결과 및 메트릭을 저장하는 데 사용됩니다. 이러한 용적은 PVC 이름을 사용하여 작업 정의에서 참조됩니다. Kubernetes 작업에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서".

   이 예시 작업이 생성하는 포드의 /dev/shm에 Memory의 midium 값을 가진 emptyDir 볼륨이 실장된다. Docker 컨테이너 런타임을 통해 자동으로 생성되는 '/dev/shm' 가상 볼륨의 기본 크기는 TensorFlow의 요구 사항에 비해 부족할 수 있습니다. 다음 예제와 같이 "emptyDir" 볼륨을 마운트하면 충분히 큰 "/dev/shm" 가상 볼륨이 제공됩니다. 'emptyDir' 볼륨에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서".

   이 예제 작업 정의에 지정된 단일 컨테이너에는 'ecurityContext > privileged' 값이 'true'로 지정됩니다. 이 값은 컨테이너가 호스트에 대한 루트 액세스 권한을 효과적으로 가지고 있음을 의미합니다. 이 경우 실행되는 특정 워크로드에 루트 액세스가 필요하므로 이 주석이 사용됩니다. 특히, 워크로드가 수행하는 명확한 캐시 작업에서는 루트 액세스가 필요합니다. 이 "특권" 주석이 필요한지 여부는 실행 중인 특정 워크로드의 요구 사항에 따라 달라집니다.

```
$ 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. 1단계에서 만든 작업이 올바르게 실행 중인지 확인합니다. 다음 명령 예에서는 작업 정의에 지정된 대로 작업에 대해 단일 POD가 생성되었으며 이 POD가 현재 GPU 작업자 노드 중 하나에서 실행되고 있음을 확인합니다.

```
$ 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. 1단계에서 생성한 작업이 성공적으로 완료되었는지 확인합니다. 다음 명령 예에서는 작업이 성공적으로 완료되었음을 확인합니다.

```
$ 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. * 선택 사항: * 작업 아티팩트 정리. 다음 예제 명령은 1단계에서 만든 작업 오브젝트의 삭제를 보여 줍니다.

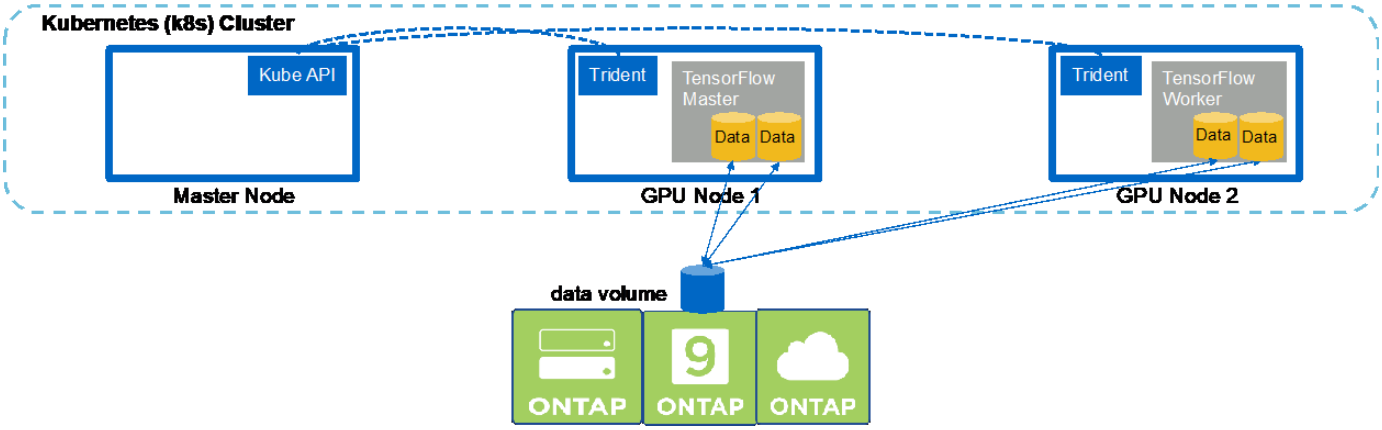
   작업 개체를 삭제하면 Kubernetes에서 연결된 포드를 자동으로 삭제합니다.

```
$ 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.
```

동기식 분산 AI 워크로드 실행

Kubernetes 클러스터에서 동기식 다중 노드 AI 및 ML 작업을 실행하려면 배포 점프 호스트에서 다음 작업을 수행하십시오. 이 프로세스를 통해 NetApp 볼륨에 저장된 데이터를 활용하고 단일 작업자 노드가 제공할 수 있는 것보다 더 많은 GPU를 사용할 수 있습니다. 동기식 분산 AI 작업을 설명하는 방법은 다음 그림을 참조하십시오.

ⓘ 동기 분산 작업은 비동기 분산 작업에 비해 성능 및 교육 정확도를 높일 수 있습니다. 동기 작업과 비동기 작업의 장단점을 논하는 것은 이 문서의 범위를 벗어납니다.



1. 다음 명령 예는 섹션의 예에서 단일 노드에서 실행된 동일한 TensorFlow 벤치마크 작업의 동기식 분산 실행에 참여하는 작업자 1명의 생성을 보여 줍니다 "단일 노드 AI 워크로드 실행". 이 특정 예제에서는 작업이 두 작업자 노드에 걸쳐 실행되므로 한 명의 작업자만 배포됩니다.

   이 작업자 배포는 8개의 GPU를 요청하므로 8개 이상의 GPU를 갖춘 단일 GPU 작업자 노드에서 실행할 수 있습니다. GPU 작업자 노드에서 8개 이상의 GPU를 사용하여 성능을 극대화한 경우, 이 숫자를 작업자 노드가 갖춘 GPU 수와 같게 늘리고 싶을 수 있습니다. Kubernetes 구축에 대한 자세한 내용은 를 참조하십시오 "Kubernetes

공식 문서".

이 예에서는 특정 컨테이너형 작업자가 자체적으로 완료되지 않기 때문에 Kubernetes 구축이 생성됩니다. 따라서 Kubernetes 작업 구성을 사용하여 구축하는 것은 타당하지 않습니다. 작업자가 혼자서 완료되도록 설계되거나 작성된 경우 작업 구성을 사용하여 작업자를 배포하는 것이 합리일 수 있습니다.

이 예제 배포 사양에 지정된 POD에 "true"의 "hostNetwork" 값이 제공됩니다. 이 값은 이 Pod가 일반적으로 Kubernetes에서 각 Pod에 생성하는 가상 네트워킹 스택 대신 호스트 작업자 노드의 네트워킹 스택을 사용한다는 것을 의미합니다. 이 경우 특정 워크로드는 개방형 MPI, NCCL 및 Horovod를 통해 동기식 분산 방식으로 워크로드를 실행하기 때문에 이 주석이 사용됩니다. 따라서 호스트 네트워킹 스택에 액세스해야 합니다. 공개 MPI, NCCL 및 Horovod에 대한 논의는 이 문서의 범위를 벗어납니다. 이 "hostNetwork: true" 주석이 필요한지 여부는 실행 중인 특정 워크로드의 요구 사항에 따라 달라집니다. hostNetwork 필드에 대한 자세한 내용은 를 참조하십시오 "Kubernetes 공식 문서".

```
$ 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. 1단계에서 만든 작업자 배포가 성공적으로 시작되었는지 확인합니다. 다음 예제 명령은 배포 정의에 나와 있는 것처럼 단일 작업자 POD가 배포용으로 생성되었으며 이 POD가 현재 GPU 작업자 노드 중 하나에서 실행되고 있음을 확인합니다.

```
$ 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. 동기 다중 노드 작업의 실행을 종료, 참여 및 추적하는 마스터에 대한 Kubernetes 작업을 생성합니다. 다음 명령 예에서는 이 섹션의 예제에서 단일 노드에서 실행된 것과 동일한 TensorFlow 벤치마크 작업의 동기식 분산 실행을 시작, 참여 및 추적하는 하나의 마스터를 생성합니다 "단일 노드 AI 워크로드 실행".

   이 마스터 작업에서는 8개의 GPU를 요청하므로 8개 이상의 GPU를 갖춘 단일 GPU 작업자 노드에서 실행할 수 있습니다. GPU 작업자 노드에서 8개 이상의 GPU를 사용하여 성능을 극대화한 경우, 이 숫자를 작업자 노드가 갖춘 GPU 수와 같게 늘리고 싶을 수 있습니다.

   이 예에서 지정한 마스터 포드는 1단계에서 작업자 포드가 hostNetwork 값이 true인 것처럼 true의 hostNetwork 값이 지정됩니다. 이 값이 필요한 이유에 대한 자세한 내용은 1단계를 참조하십시오.

```
$ 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. 3단계에서 만든 마스터 작업이 올바르게 실행되고 있는지 확인합니다. 다음 예제 명령은 작업 정의에 나와 있는 것처럼 작업에 대해 단일 마스터 포드가 생성되었으며 이 포드가 현재 GPU 작업자 노드 중 하나에서 실행되고 있음을 확인합니다. 또한 1단계에서 처음 보았던 작업자 포드가 여전히 실행 중이고 마스터 포드와 작업자 포드가 다른 노드에서 실행되고 있음을 확인해야 합니다.

```
$ 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. 3단계에서 만든 마스터 작업이 성공적으로 완료되었는지 확인합니다. 다음 명령 예에서는 작업이 성공적으로 완료되었음을 확인합니다.

```
$ 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. 작업자 배포가 더 이상 필요하지 않으면 삭제합니다. 다음 예제 명령은 1단계에서 만든 작업자 배포 개체를 삭제하는 방법을 보여 줍니다.

   작업자 배포 개체를 삭제하면 Kubernetes에서 연결된 작업자 포드를 자동으로 삭제합니다.

```
$ 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. * 선택 사항: * 마스터 작업 아티팩트를 정리하십시오. 다음 예제 명령은 3단계에서 만든 마스터 작업 오브젝트의 삭제를 보여 줍니다.

   마스터 작업 개체를 삭제하면 연결된 마스터 포드가 자동으로 삭제됩니다.

```
$ 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.
```