NetApp 사용한 오픈 소스 MLOps

NetApp artificial intelligence solutions

NetApp
February 12, 2026

목차

NetApp 사용한 오픈 소스 MLOps

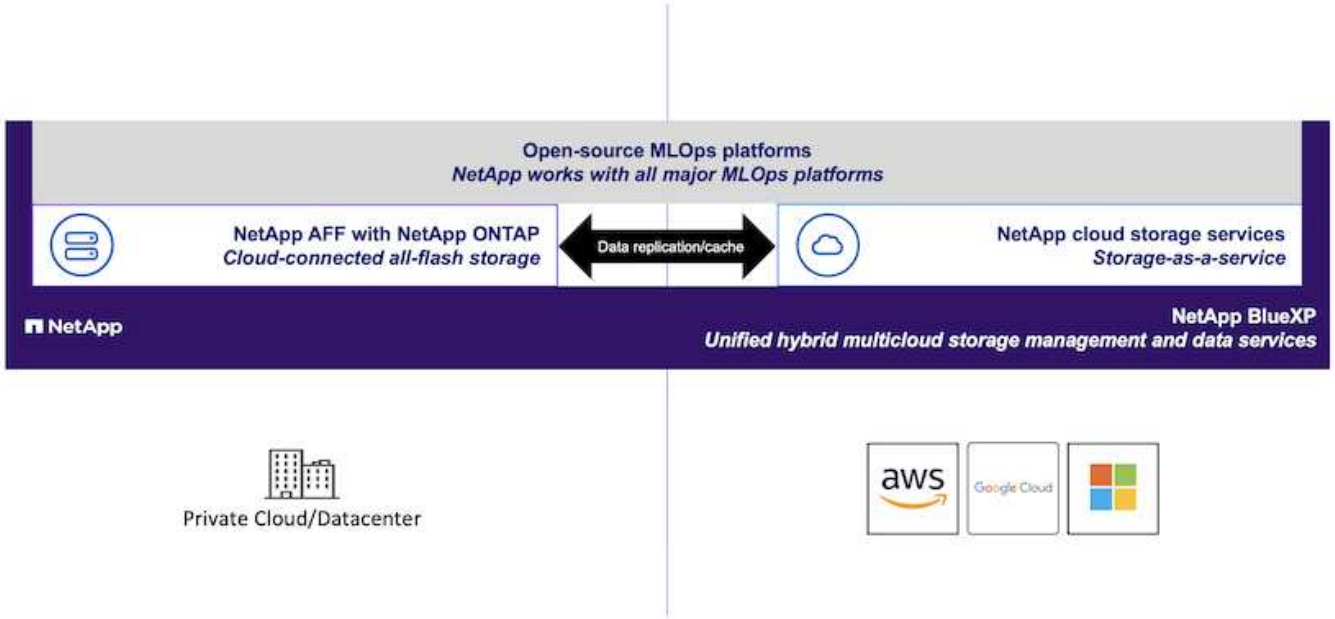
NetApp 사용한 오픈 소스 MLOps

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

다양한 산업 분야의 모든 규모의 회사와 조직은 실제 문제를 해결하고, 혁신적인 제품과 서비스를 제공하며, 점점 경쟁이 치열해지는 시장에서 우위를 점하기 위해 인공지능(AI)에 눈을 돌리고 있습니다. 많은 조직이 업계의 빠른 혁신 속도에 발맞추기 위해 오픈소스 MLOps 도구를 선택하고 있습니다. 이러한 오픈소스 도구는 고급 기능과 최첨단 기능을 제공하지만, 데이터 가용성과 데이터 보안을 고려하지 않는 경우가 많습니다. 안타깝게도 이는 고도로 숙련된 데이터 과학자들이 데이터에 액세스하거나 기본적인 데이터 관련 작업이 완료될 때까지 기다리는 데 상당한 시간을 소비해야 한다는 것을 의미합니다. 인기 있는 오픈소스 MLOps 도구를 NetApp 의 지능형 데이터 인프라와 결합하면 조직은 데이터 파이프라인을 가속화할 수 있으며, 이를 통해 AI 이니셔티브도 가속화됩니다. 데이터가 보호되고 보안되는 것을 보장하는 동시에 데이터에서 가치를 창출할 수 있습니다. 이 솔루션은 이러한 과제를 해결하기 위해 NetApp 데이터 관리 기능과 여러 가지 인기 있는 오픈 소스 도구 및 프레임워크를 결합하는 방법을 보여줍니다.

다음 목록은 이 솔루션을 통해 구현되는 몇 가지 주요 기능을 강조합니다.

- 사용자는 고성능, 확장형 NetApp 스토리지를 기반으로 새로운 대용량 데이터 볼륨과 개발 작업 공간을 빠르게 프로비저닝할 수 있습니다.
- 사용자는 대용량 데이터 볼륨과 개발 작업 공간을 거의 즉각적으로 복제하여 실험이나 빠른 반복을 수행할 수 있습니다.
- 사용자는 대용량 데이터 볼륨과 개발 작업 공간의 스냅샷을 거의 즉시 저장하여 백업 및/또는 추적/기준 설정을 수행할 수 있습니다.



일반적인 MLOps 워크플로는 일반적으로 다음과 같은 형태를 갖는 개발 작업 공간을 통합합니다."주피터 노트북" ; 실험 추적; 자동화된 교육 파이프라인; 데이터 파이프라인; 추론/배포. 이 솔루션은 워크플로의 다양한 측면을 해결하기 위해

독립적으로 또는 함께 사용할 수 있는 여러 가지 도구와 프레임워크를 강조합니다. 또한 NetApp 데이터 관리 기능과 각 도구의 페어링을 보여드립니다. 이 솔루션은 조직이 자사의 사용 사례와 요구 사항에 맞춰 맞춤형 MLOps 워크플로를 구성할 수 있는 기본 요소를 제공하기 위해 고안되었습니다.

이 솔루션에는 다음과 같은 도구/프레임워크가 포함되어 있습니다.

- "아파치 에어플로우"
- "주피터허브"
- "쿠베플로우"
- "ML플로우"

다음 목록은 이러한 도구를 독립적으로 또는 결합하여 배포하는 일반적인 패턴을 설명합니다.

- JupyterHub, MLflow 및 Apache Airflow를 함께 배포 - JupyterHub"주피터 노트북" 실험 추적을 위한 MLflow, 자동화된 학습 및 데이터 파이프라인을 위한 Apache Airflow.
- Kubeflow와 Apache Airflow를 함께 배포 - Kubeflow for"주피터 노트북" , 실험 추적, 자동화된 교육 파이프라인 및 추론, 데이터 파이프라인을 위한 Apache Airflow.
- Kubeflow를 올인원 MLOps 플랫폼 솔루션으로 배포"주피터 노트북" , 실험 추적, 자동화된 학습 및 데이터 파이프라인, 추론.

기술 개요

이 섹션에서는 NetApp 사용한 오픈소스 MLOps에 대한 기술 개요에 중점을 둡니다.

인공지능

AI는 컴퓨터가 인간 정신의 인지 기능을 모방하도록 훈련되는 컴퓨터 과학 분야입니다. AI 개발자는 컴퓨터가 인간과 비슷하거나 더 나은 방식으로 학습하고 문제를 해결하도록 훈련시킵니다. 딥러닝과 머신러닝은 AI의 하위 분야입니다. 점점 더 많은 기업이 중요한 비즈니스 요구 사항을 지원하기 위해 AI, ML, DL을 도입하고 있습니다. 몇 가지 예는 다음과 같습니다.

- 이전에 알려지지 않았던 비즈니스 통찰력을 발굴하기 위해 방대한 양의 데이터 분석
- 자연어 처리를 사용하여 고객과 직접 상호 작용
- 다양한 비즈니스 프로세스 및 기능 자동화

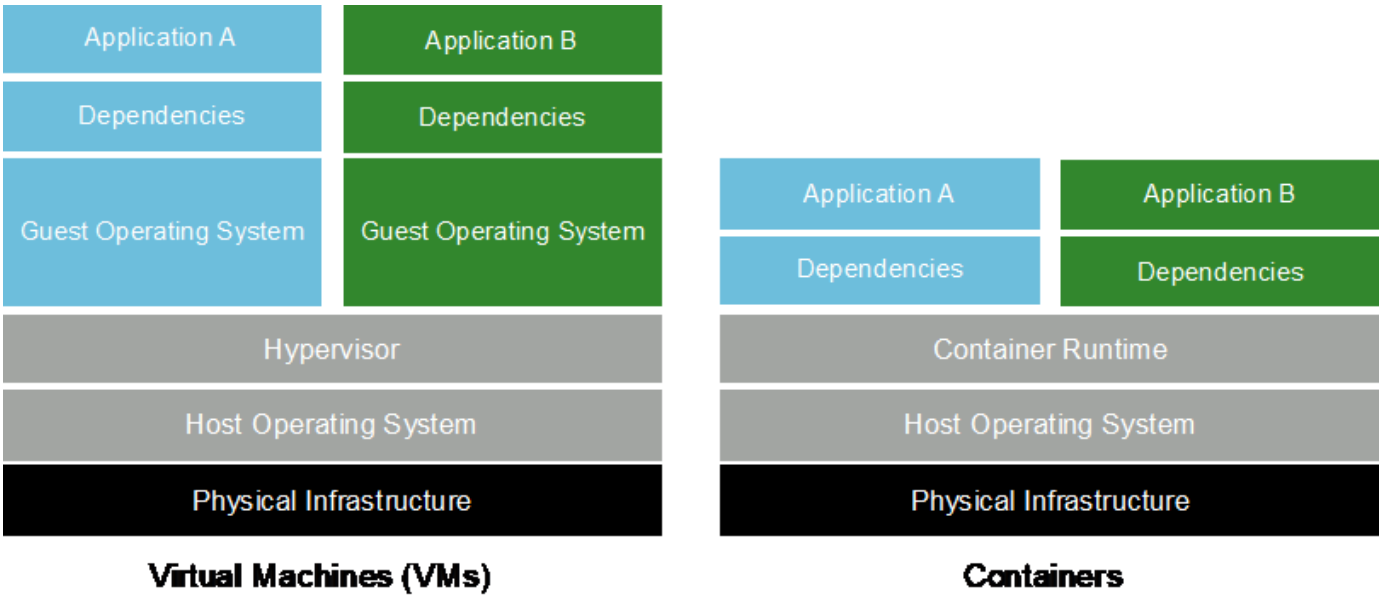
최신 AI 학습 및 추론 작업에는 대규모 병렬 컴퓨팅 기능이 필요합니다. 따라서 GPU는 AI 작업을 실행하는 데 점점 더 많이 사용되고 있는데, 그 이유는 GPU의 병렬 처리 능력이 범용 CPU보다 훨씬 뛰어나기 때문입니다.

컨테이너

컨테이너는 공유 호스트 운영 체제 커널 위에서 실행되는 격리된 사용자 공간 인스턴스입니다. 컨테이너의 도입이 급속히 증가하고 있습니다. 컨테이너는 가상 머신(VM)이 제공하는 것과 동일한 애플리케이션 샌드박싱 이점을 많이 제공합니다. 하지만 VM이 의존하는 하이퍼바이저와 게스트 운영 체제 계층이 제거되었기 때문에 컨테이너는 훨씬 더 가볍습니다. 다음 그림은 가상 머신과 컨테이너를 시각화하여 보여줍니다.

컨테이너를 사용하면 애플리케이션 종속성, 런타임 등을 애플리케이션과 직접 효율적으로 패키징할 수도 있습니다. 가장 일반적으로 사용되는 컨테이너 패키징 형식은 Docker 컨테이너입니다. Docker 컨테이너 형식으로 컨테이너화된 애플리케이션은 Docker 컨테이너를 실행할 수 있는 모든 머신에서 실행될 수 있습니다. 이는 애플리케이션의 종속성이

머신에 존재하지 않더라도 사실입니다. 모든 종속성은 컨테이너 자체에 패키징되어 있기 때문입니다. 자세한 내용은 다음을 방문하세요. "Docker 웹사이트" .



쿠버네티스

쿠버네티스는 원래 Google에서 설계한 오픈 소스, 분산형 컨테이너 오케스트레이션 플랫폼으로, 현재는 Cloud Native Computing Foundation(CNCF)에서 관리하고 있습니다. 쿠버네티스는 컨테이너화된 애플리케이션의 배포, 관리 및 확장 기능을 자동화할 수 있습니다. 최근 몇 년 동안 쿠버네티스는 지배적인 컨테이너 오케스트레이션 플랫폼으로 떠올랐습니다. 자세한 내용은 다음을 방문하세요. "쿠버네티스 웹사이트" .

NetApp Trident

"Trident"ONTAP (AFF, FAS, Select, Cloud, Amazon FSx ONTAP), Azure NetApp Files 서비스, Google Cloud NetApp Volumes 포함하여 퍼블릭 클라우드 또는 온프레미스에서 모든 인기 있는 NetApp 스토리지 플랫폼에서 스토리지 리소스를 사용하고 관리할 수 있도록 지원합니다. Trident 는 Kubernetes와 기본적으로 통합되는 컨테이너 스토리지 인터페이스(CSI) 호환 동적 스토리지 오케스트레이터입니다.

NetApp DataOps 툴킷

그만큼"NetApp DataOps 툴킷" 고성능, 확장형 NetApp 스토리지에 의해 지원되는 개발/교육 작업 공간과 추론 서버의 관리를 간소화하는 Python 기반 도구입니다. 주요 기능은 다음과 같습니다.

- 고성능, 확장형 NetApp 스토리지로 지원되는 대용량의 새로운 작업 공간을 빠르게 프로비저닝하세요.
- 실험이나 빠른 반복을 가능하게 하기 위해 대용량 작업 공간을 거의 즉각적으로 복제합니다.
- 대용량 작업 공간의 스냅샷을 거의 즉각적으로 저장하여 백업 및/또는 추적/기준 설정을 수행합니다.
- 거의 즉각적으로 대용량, 고성능 데이터 볼륨을 프로비저닝, 복제 및 스냅샷합니다.

아파치 에어플로우

Apache Airflow는 복잡한 엔터프라이즈 워크플로에 대한 프로그래밍 방식 작성, 일정 예약 및 모니터링을 지원하는 오픈 소스 워크플로 관리 플랫폼입니다. ETL 및 데이터 파이프라인 워크플로를 자동화하는 데 자주 사용되지만 이러한 유형의 워크플로에만 국한되지는 않습니다. Airflow 프로젝트는 Airbnb에서 시작되었지만 그 이후로 업계에서 큰 인기를

얻었으며 현재는 Apache 소프트웨어 재단의 후원을 받고 있습니다. Airflow는 Python으로 작성되었으며, Airflow 워크플로는 Python 스크립트를 통해 생성되고, Airflow는 "코드로 구성"이라는 원칙에 따라 설계되었습니다. 많은 기업 Airflow 사용자는 이제 Kubernetes를 기반으로 Airflow를 실행합니다.

방향성 비순환 그래프(DAG)

Airflow에서는 워크플로를 DAG(Directed Acycle Graphs)라고 합니다. DAG는 DAG 정의에 따라 순차적으로, 병렬로 또는 두 가지를 조합하여 실행되는 작업으로 구성됩니다. Airflow 스케줄러는 DAG 정의에 지정된 작업 수준 종속성을 준수하여 일련의 작업자에서 개별 작업을 실행합니다. DAG는 Python 스크립트를 통해 정의되고 생성됩니다.

주피터 노트북

Jupyter Notebook은 실시간 코드와 설명 텍스트를 포함하는 위키와 유사한 문서입니다. Jupyter Notebooks는 AI 및 ML 커뮤니티에서 AI 및 ML 프로젝트를 문서화하고, 저장하고, 공유하는 수단으로 널리 사용됩니다. Jupyter Notebooks에 대한 자세한 내용은 다음을 방문하세요. "주피터 웹사이트" .

Jupyter Notebook 서버

Jupyter Notebook Server는 사용자가 Jupyter Notebook을 만들 수 있는 오픈 소스 웹 애플리케이션입니다.

주피터허브

JupyterHub는 개별 사용자가 자신의 Jupyter Notebook 서버를 프로비저닝하고 액세스할 수 있도록 하는 다중 사용자 애플리케이션입니다. JupyterHub에 대한 자세한 내용은 다음을 방문하세요. "JupyterHub 웹사이트" .

ML플로우

MLflow는 인기 있는 오픈소스 AI 라이프사이클 관리 플랫폼입니다. MLflow의 주요 기능으로는 AI/ML 실험 추적 및 AI/ML 모델 저장소가 있습니다. MLflow에 대한 자세한 내용은 다음을 방문하세요. "MLflow 웹사이트" .

쿠베플로우

Kubeflow는 원래 Google에서 개발한 Kubernetes용 오픈소스 AI 및 ML 툴킷입니다. Kubeflow 프로젝트는 Kubernetes에서 AI 및 ML 워크플로를 간편하고, 이식 가능하며, 확장 가능하게 배포할 수 있도록 해줍니다. Kubeflow는 Kubernetes의 복잡한 부분을 추상화하여 데이터 과학자가 자신이 가장 잘 아는 분야인 데이터 과학에 집중할 수 있도록 해줍니다. 다음 그림을 통해 시각화를 살펴보세요. Kubeflow는 올인원 MLOps 플랫폼을 선호하는 조직에 적합한 오픈소스 옵션입니다. 자세한 내용은 다음을 방문하세요. "Kubeflow 웹사이트" .

Kubeflow 파이프라인

Kubeflow 파이프라인은 Kubeflow의 핵심 구성 요소입니다. Kubeflow Pipelines는 이식 가능하고 확장 가능한 AI 및 ML 워크플로를 정의하고 배포하기 위한 플랫폼이자 표준입니다. 자세한 내용은 다음을 참조하세요. "공식 Kubeflow 문서" .

Kubeflow 노트북

Kubeflow는 Kubernetes에서 Jupyter Notebook 서버의 프로비저닝과 배포를 간소화합니다. Kubeflow 컨텍스트 내에서 Jupyter Notebooks에 대한 자세한 내용은 다음을 참조하세요. "공식 Kubeflow 문서" .

카티브

Katib은 자동화된 머신 러닝(AutoML)을 위한 Kubernetes 기반 프로젝트입니다. Katib은 하이퍼파라미터 튜닝, 조기 중단 및 NAS(신경망 구조 탐색)를 지원합니다. Katib은 머신 러닝(ML) 프레임워크에 구애받지 않는 프로젝트입니다. 사용자가 선택한 언어로 작성된 애플리케이션의 하이퍼파라미터를 조정할 수 있으며 TensorFlow, MXNet, PyTorch, XGBoost 등 다양한 ML 프레임워크를 기본적으로 지원합니다. Katib은 베이지안 최적화, 파젠 추정 트리, 무작위 탐색, 공분산 행렬 적응 진화 전략, 하이퍼밴드, 효율적 신경 구조 탐색, 미분 가능 구조 탐색 등 다양한 AutoML 알고리즘을 지원합니다. Kubeflow 컨텍스트 내에서 Jupyter Notebooks에 대한 자세한 내용은 다음을 참조하세요. "공식 Kubeflow 문서" .

NetApp ONTAP

NetApp 의 최신 스토리지 관리 소프트웨어인 ONTAP 9를 사용하면 기업이 인프라를 현대화하고 클라우드 지원 데이터 센터로 전환할 수 있습니다. ONTAP 업계 최고의 데이터 관리 역량을 활용하여 데이터가 어디에 있든 단일 도구 세트를 사용하여 데이터를 관리하고 보호할 수 있도록 지원합니다. 또한 필요한 곳, 즉 엣지, 코어, 클라우드로 데이터를 자유롭게 이동할 수 있습니다. ONTAP 9에는 데이터 관리를 간소화하고, 중요 데이터를 가속화하고 보호하며, 하이브리드 클라우드 아키텍처 전반에서 차세대 인프라 기능을 구현하는 다양한 기능이 포함되어 있습니다.

데이터 관리 간소화

적절한 리소스가 AI 애플리케이션과 AI/ML 데이터 세트 교육에 사용될 수 있도록 기업 IT 운영과 데이터 과학자에게 데이터 관리가 매우 중요합니다. NetApp 기술에 대한 다음 추가 정보는 이 검증 범위를 벗어나지만 배포에 따라 관련이 있을 수 있습니다.

ONTAP 데이터 관리 소프트웨어에는 다음과 같은 기능이 포함되어 있어 운영을 간소화하고 단순화하며 총 운영 비용을 절감할 수 있습니다.

- 인라인 데이터 압축 및 확장된 중복 제거. 데이터 압축은 저장 블록 내부의 낭비되는 공간을 줄이고, 중복 제거는 효과적인 용량을 크게 증가시킵니다. 이는 로컬에 저장된 데이터와 클라우드에 계층화된 데이터 모두에 적용됩니다.
- 최소, 최대 및 적응형 서비스 품질(AQoS). 세분화된 서비스 품질(QoS) 제어는 공유 빈도가 높은 환경에서 중요한 애플리케이션의 성능 수준을 유지하는 데 도움이 됩니다.
- NetApp FabricPool. Amazon Web Services(AWS), Azure, NetApp StorageGRID 스토리지 솔루션을 포함한 퍼블릭 및 프라이빗 클라우드 스토리지 옵션에 콜드 데이터의 자동 계층화를 제공합니다. FabricPool 에 대한 자세한 내용은 다음을 참조하세요. "TR-4598: FabricPool 모범 사례" .

데이터 가속화 및 보호

ONTAP 뛰어난 수준의 성능과 데이터 보호 기능을 제공하며 다음과 같은 방식으로 이러한 기능을 확장합니다.

- 성능과 낮은 지연 시간. ONTAP 가능한 가장 낮은 지연 시간으로 가능한 가장 높은 처리량을 제공합니다.
- 데이터 보호. ONTAP 모든 플랫폼에서 공통적으로 관리할 수 있는 내장형 데이터 보호 기능을 제공합니다.
- NetApp 볼륨 암호화(NVE). ONTAP 온보드 및 외부 키 관리 지원을 통해 기본 볼륨 수준 암호화를 제공합니다.
- 다중 테넌시 및 다중 요소 인증. ONTAP 최고 수준의 보안을 통해 인프라 리소스를 공유할 수 있도록 합니다.

미래 지향적 인프라

ONTAP 다음과 같은 기능을 통해 까다롭고 끊임없이 변화하는 비즈니스 요구 사항을 충족하는 데 도움이 됩니다.

- 원활한 확장과 중단 없는 운영. ONTAP 기존 컨트롤러와 확장형 클러스터에 중단 없이 용량을 추가할 수 있도록 지원합니다. 고객은 비용이 많이 드는 데이터 마이그레이션이나 중단 없이 최신 기술로 업그레이드할 수 있습니다.

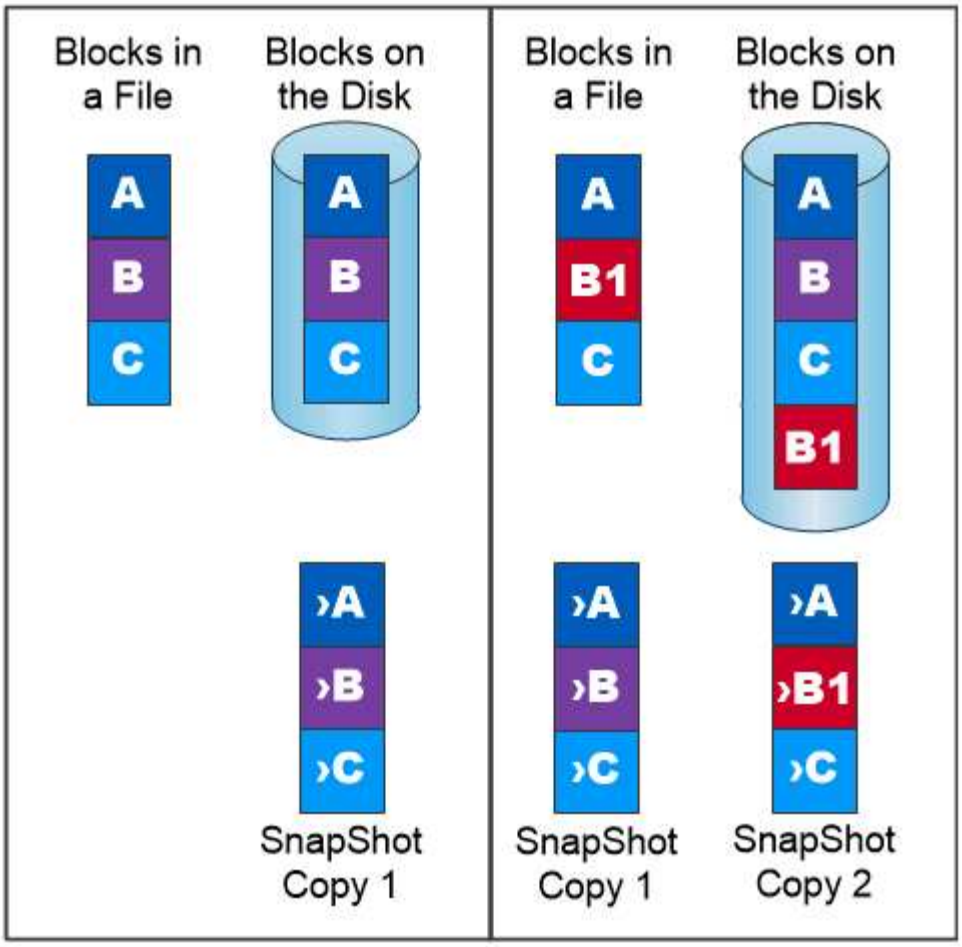
- 클라우드 연결. ONTAP 은 모든 퍼블릭 클라우드에서 소프트웨어 정의 스토리지와 클라우드 네이티브 인스턴스에 대한 옵션을 갖춘 가장 클라우드에 연결된 스토리지 관리 소프트웨어입니다.
- 새로운 애플리케이션과의 통합. ONTAP 기존 엔터프라이즈 앱을 지원하는 동일한 인프라를 사용하여 자율주행차, 스마트 시티, 산업 4.0과 같은 차세대 플랫폼과 애플리케이션을 위한 엔터프라이즈급 데이터 서비스를 제공합니다.

NetApp 스냅샷 복사본

NetApp 스냅샷 복사본은 볼륨의 읽기 전용 특정 시점 이미지입니다. 다음 그림에서 볼 수 있듯이 이미지는 최소한의 저장 공간을 사용하고 마지막 스냅샷 복사본이 만들어진 이후에 생성된 파일의 변경 사항만 기록하므로 성능 오버헤드가 무시할 수 있을 정도입니다.

스냅샷 복사본의 효율성은 핵심 ONTAP 스토리지 가상화 기술인 WAFL(Write Anywhere File Layout) 덕분에 가능합니다. WAFL 데이터베이스와 마찬가지로 메타데이터를 사용하여 디스크의 실제 데이터 블록을 가리킵니다. 하지만 데이터베이스와 달리 WAFL 기존 블록을 덮어쓰지 않습니다. 업데이트된 데이터를 새로운 블록에 쓰고 메타데이터를 변경합니다. ONTAP 스냅샷 복사본을 생성할 때 데이터 블록을 복사하는 대신 메타데이터를 참조하기 때문에 스냅샷 복사본이 매우 효율적입니다. 그렇게 하면 다른 시스템이 복사할 블록을 찾는 데 걸리는 탐색 시간과 복사 자체를 만드는 데 드는 비용을 없앨 수 있습니다.

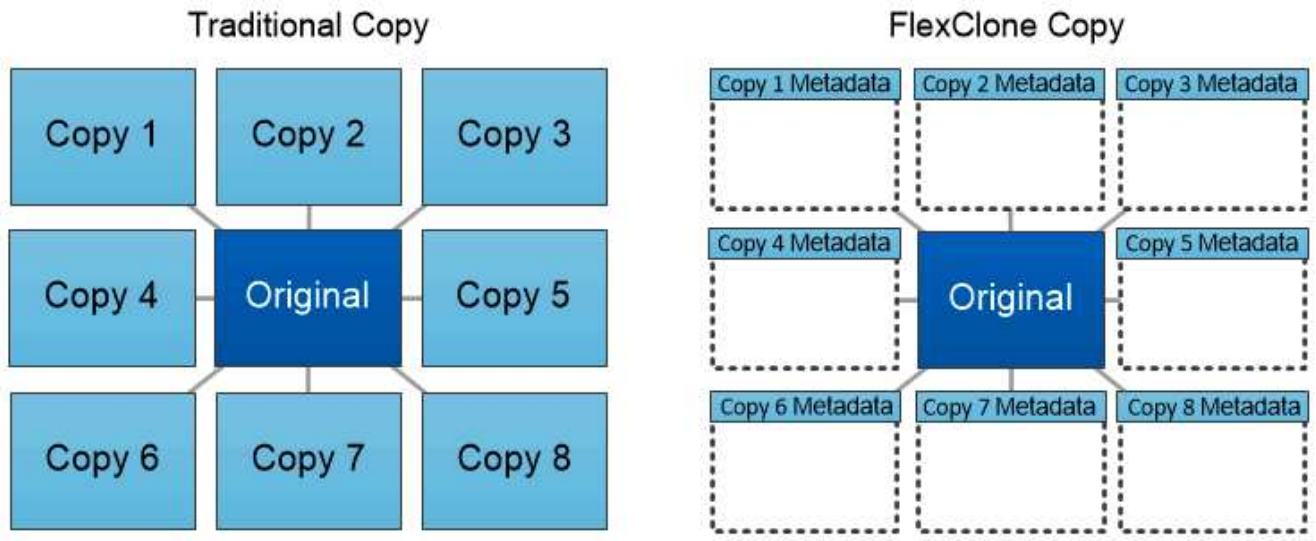
스냅샷 복사본을 사용하면 개별 파일이나 LUN을 복구하거나 볼륨의 전체 내용을 복원할 수 있습니다. ONTAP 스냅샷 복사본의 포인터 정보를 디스크의 데이터와 비교하여 다운타임이나 상당한 성능 비용 없이 누락되거나 손상된 객체를 재구성합니다.



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

NetApp FlexClone 기술

NetApp FlexClone 기술은 스냅샷 메타데이터를 참조하여 볼륨의 쓰기 가능한 특정 시점 복사본을 생성합니다. 다음 그림에서 볼 수 있듯이 복사본은 부모와 데이터 블록을 공유하며, 복사본에 변경 사항이 기록될 때까지 메타데이터에 필요한 것을 제외하고는 저장 공간을 사용하지 않습니다. 기존 복사 작업에는 몇 분 또는 몇 시간이 걸릴 수 있지만, FlexClone 소프트웨어를 사용하면 가장 큰 데이터 세트도 거의 즉시 복사할 수 있습니다. 따라서 동일한 데이터 세트의 여러 사본이 필요한 상황(예: 개발 작업 공간)이나 데이터 세트의 임시 사본(예: 프로덕션 데이터 세트에 대한 애플리케이션 테스트)이 필요한 경우에 이상적입니다.



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 데이터 패브릭 전반에 걸친 비용 효율적이고 사용하기 쉬운 통합 복제 솔루션입니다. LAN이나 WAN을 통해 고속으로 데이터를 복제합니다. 이 솔루션은 가상 및 기존 환경 모두에서 비즈니스에 중요한 애플리케이션을 포함하여 모든 유형의 애플리케이션에 대해 높은 데이터 가용성과 빠른 데이터 복제 기능을 제공합니다. 하나 이상의 NetApp 스토리지 시스템에 데이터를 복제하고 보조 데이터를 지속적으로 업데이트하면 데이터가 최신 상태로 유지되고 필요할 때마다 사용할 수 있습니다. 외부 복제 서버가 필요하지 않습니다. SnapMirror 기술을 활용하는 아키텍처의 예는 다음 그림을 참조하세요.

SnapMirror 소프트웨어는 변경된 블록만 네트워크를 통해 전송하여 NetApp ONTAP 스토리지 효율성을 활용합니다. SnapMirror 소프트웨어는 내장된 네트워크 압축 기능을 사용하여 데이터 전송 속도를 높이고 네트워크 대역폭 사용량을 최대 70%까지 줄입니다. SnapMirror 기술을 사용하면 단일 씬 복제 데이터 스트림을 활용하여 활성 미러와 이전 시점 복사본을 모두 유지하는 단일 저장소를 만들어 네트워크 트래픽을 최대 50%까지 줄일 수 있습니다.

NetApp BlueXP 복사 및 동기화

"BlueXP 복사 및 동기화"빠르고 안전한 데이터 동기화를 위한 NetApp 서비스입니다. 온프레미스 NFS 또는 SMB 파일 공유, NetApp StorageGRID, NetApp ONTAP S3, Google Cloud NetApp Volumes, Azure NetApp Files, AWS S3, AWS EFS, Azure Blob, Google Cloud Storage 또는 IBM Cloud Object Storage 간에 파일을 전송해야 하는 경우 BlueXP Copy and Sync를 사용하면 파일을 필요한 곳으로 빠르고 안전하게 이동할 수 있습니다.

데이터가 전송되면 소스와 타겟 모두에서 자유롭게 사용할 수 있습니다. BlueXP Copy and Sync는 업데이트가 발생할 때 필요에 따라 데이터를 동기화하거나 미리 정의된 일정에 따라 지속적으로 데이터를 동기화할 수 있습니다. 그럼에도 불구하고 BlueXP Copy and Sync는 델타만 이동하므로 데이터 복제에 소요되는 시간과 비용이 최소화됩니다.

BlueXP Copy and Sync는 설정과 사용이 매우 간단한 SaaS(Software as a Service) 도구입니다. BlueXP Copy and Sync에 의해 트리거되는 데이터 전송은 데이터 브로커를 통해 수행됩니다. BlueXP 복사 및 동기화 데이터 브로커는 AWS, Azure, Google Cloud Platform 또는 온프레미스에 배포할 수 있습니다.

NetApp XCP

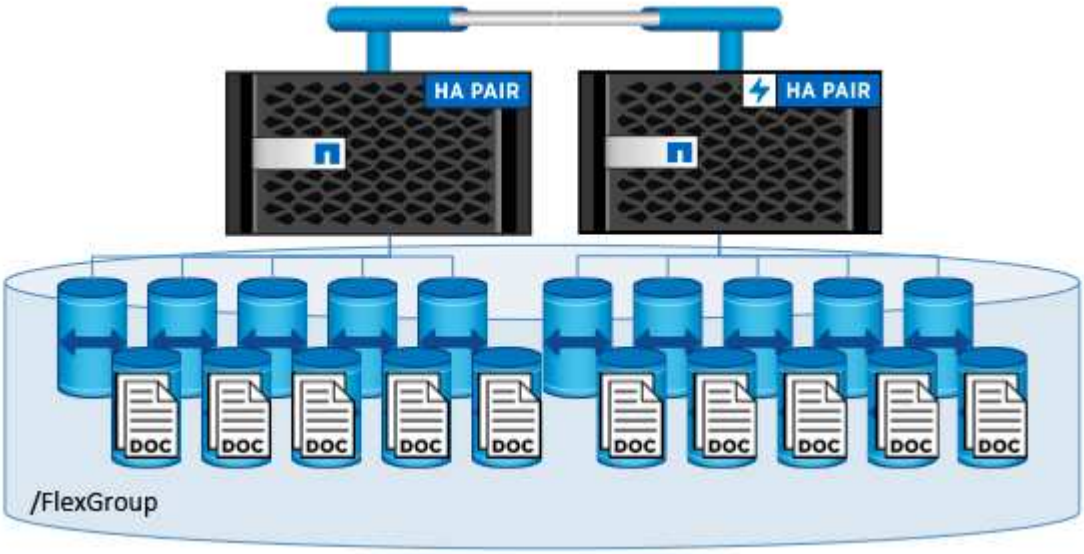
"NetApp XCP"모든 NetApp 및 NetApp NetApp 데이터 마이그레이션과 파일 시스템 통찰력을 위한 클라이언트 기반 소프트웨어입니다. XCP는 사용 가능한 모든 시스템 리소스를 활용해 대용량 데이터 세트와 고성능 마이그레이션을 처리함으로써 확장성을 높이고 최대 성능을 달성하도록 설계되었습니다. XCP는 보고서 생성 옵션을 통해 파일 시스템에 대한 완전한 가시성을 확보하는 데 도움이 됩니다.

NetApp ONTAP FlexGroup 볼륨

훈련 데이터 세트는 잠재적으로 수십억 개의 파일 모음이 될 수 있습니다. 파일에는 텍스트, 오디오, 비디오 및 기타 형태의 비정형 데이터가 포함될 수 있으며, 이러한 데이터는 병렬로 읽을 수 있도록 저장하고 처리해야 합니다. 저장 시스템은 많은 수의 작은 파일을 저장해야 하며, 순차적이고 무작위적인 I/O를 위해 해당 파일을 병렬로 읽어야 합니다.

FlexGroup 볼륨은 다음 그림에서 볼 수 있듯이 여러 구성 멤버 볼륨으로 구성된 단일 네임스페이스입니다. 스토리지 관리자 관점에서 FlexGroup 볼륨은 NetApp FlexVol volume 처럼 관리되고 작동합니다. FlexGroup 볼륨의 파일은 개별 멤버 볼륨에 할당되며 볼륨이나 노드에 걸쳐 스트라이프되지 않습니다. 다음과 같은 기능을 제공합니다.

- FlexGroup 볼륨은 대량의 메타데이터 워크로드에 대해 수 페타바이트의 용량과 예측 가능한 낮은 대기 시간을 제공합니다.
- 동일한 네임스페이스에서 최대 4,000억 개의 파일을 지원합니다.
- 이들은 CPU, 노드, 집계 및 구성 FlexVol 볼륨 전반의 NAS 워크로드에서 병렬화된 작업을 지원합니다.



아키텍처

이 솔루션은 특정 하드웨어에 의존하지 않습니다. 이 솔루션은 NetApp Trident 가 지원하는 모든 NetApp 물리적 스토리지 어플라이언스, 소프트웨어 정의 인스턴스 또는 클라우드 서비스와 호환됩니다. 예로는 NetApp AFF 스토리지 시스템, Amazon FSx ONTAP, Azure NetApp Files, Google Cloud NetApp Volumes 또는 NetApp Cloud Volumes ONTAP 인스턴스가 있습니다. 또한, NetApp Trident 와 구현 중인 다른 솔루션 구성 요소에서 지원되는 Kubernetes 버전을 사용하는 한 모든 Kubernetes 클러스터에서 솔루션을 구현할 수 있습니다. Trident 에서 지원하는 Kubernetes 버전 목록은 다음을 참조하세요. "Trident 문서" . 이 솔루션의 다양한 구성

요소를 검증하는 데 사용된 환경에 대한 자세한 내용은 다음 표를 참조하세요.

Apache Airflow 검증 환경

소프트웨어 구성 요소	버전
아파치 에어플로우	2.0.1, 다음을 통해 배포됨"Apache Airflow Helm 차트" 8.0.8
쿠버네티스	1.18
NetApp Trident	21.01

JupyterHub 검증 환경

소프트웨어 구성 요소	버전
주피터허브	4.1.5, 배포됨"JupyterHub Helm 차트" 3.3.7
쿠버네티스	1.29
NetApp Trident	24.02

MLflow 검증 환경

소프트웨어 구성 요소	버전
ML플로우	2.14.1, 다음을 통해 배포됨"MLflow Helm 차트" 1.4.12
쿠버네티스	1.29
NetApp Trident	24.02

Kubeflow 검증 환경

소프트웨어 구성 요소	버전
쿠베플로우	1.7, 배포를 통해"배포KF" 0.1.1
쿠버네티스	1.26
NetApp Trident	23.07

지원하다

NetApp Apache Airflow, JupyterHub, MLflow, Kubeflow 또는 Kubernetes에 대한 엔터프라이즈 지원을 제공하지 않습니다. 완벽하게 지원되는 MLOps 플랫폼에 관심이 있으시다면,"NetApp 에 문의하세요" NetApp 파트너와 함께 제공하는 완벽하게 지원되는 MLOps 솔루션에 대해 알아보세요.

NetApp Trident 구성

NetApp AIPod 배포를 위한 Trident 백엔드 예시

Kubernetes 클러스터 내에서 Trident 사용하여 스토리지 리소스를 동적으로 프로비저닝하려면 먼저 하나 이상의 Trident 백엔드를 만들어야 합니다. 다음 예는 이 솔루션의 구성 요소를 배포하는 경우 생성하려는 다양한 유형의 백엔드를 나타냅니다."NetApp AIPod" . 백엔드에 대한 자세한 내용과 예를 들어 다른 플랫폼/환경의 백엔드에 대한 내용은 다음을 참조하세요."Trident 문서" .

1. NetApp AIPod 에 대해 FlexGroup 지원 Trident 백엔드를 만드는 것을 권장합니다.

   다음 예제 명령은 AIPod 스토리지 가상 머신(SVM)에 대한 FlexGroup 지원 Trident 백엔드를 만드는 방법을 보여줍니다. 이 백엔드는 다음을 사용합니다. `ontap-nas-flexgroup` 저장 드라이버. ONTAP FlexVol 과 FlexGroup 이라는 두 가지 주요 데이터 볼륨 유형을 지원합니다. FlexVol 볼륨은 크기가 제한되어 있습니다(이 글을 쓰는 시점에서 최대 크기는 특정 배포에 따라 달라집니다). 반면 FlexGroup 볼륨은 최대 20PB와 4,000억 개의 파일까지 선형적으로 확장할 수 있어 데이터 관리를 크게 간소화하는 단일 네임스페이스를 제공합니다. 따라서 FlexGroup 볼륨은 대량의 데이터에 의존하는 AI 및 ML 워크로드에 최적화되어 있습니다.

   소량의 데이터로 작업하고 FlexGroup 볼륨 대신 FlexVol 볼륨을 사용하려는 경우 다음을 사용하는 Trident 백엔드를 생성할 수 있습니다. `ontap-nas` 대신 저장 드라이버 `ontap-nas-flexgroup` 저장 드라이버.

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

2. NetApp FlexVol 이 지원되는 Trident 백엔드를 만드는 것도 권장합니다. 영구적인 애플리케이션을 호스팅하고, 결과, 출력, 디버그 정보 등을 저장하는 데 FlexVol 볼륨을 사용할 수 있습니다. FlexVol 볼륨을 사용하려면 FlexVol 이 활성화된 Trident 백엔드를 하나 이상 만들어야 합니다. 다음 명령 예시는 FlexVol 이 활성화된 단일 Trident 백엔드를 만드는 방법을 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 배포를 위한 Kubernetes StorageClass 예시

Kubernetes 클러스터 내에서 Trident 사용하여 스토리지 리소스를 동적으로 프로비저닝하려면 먼저 하나 이상의 Kubernetes StorageClass를 만들어야 합니다. 다음 예는 이 솔루션의 구성 요소를 배포하는 경우 생성하려는 다양한 유형의 StorageClass를 나타냅니다."NetApp AIPod" . StorageClass에 대한 자세한 내용과 다른 플랫폼/환경에 대한 StorageClass에 대한 내용은

다음을 참조하세요."Trident 문서" .

1. NetApp 섹션에서 생성한 FlexGroup 지원 Trident 백엔드에 대한 StorageClass를 생성하는 것을 권장합니다 ."NetApp AIPod 배포를 위한 Trident 백엔드 예시" , 1단계. 다음 예제 명령은 섹션에서 생성된 예제 백엔드에 해당하는 여러 StorageClass를 생성하는 방법을 보여줍니다."NetApp AIPod 배포를 위한 Trident 백엔드 예시" , 1단계 - 활용하는 단계"RDMA를 통한 NFS" 그리고 그렇지 않은 것도 하나 있습니다.

   해당 PersistentVolumeClaim(PVC)이 삭제될 때 영구 볼륨이 삭제되지 않도록 다음 예제에서는 다음을 사용합니다. `reclaimPolicy` 의 가치 `Retain` . 자세한 내용은 `reclaimPolicy` 필드, 공식을 참조하세요 "쿠버네티스 문서" .

   참고: 다음 예제 StorageClass는 최대 전송 크기인 262144를 사용합니다. 이 최대 전송 크기를 사용하려면 ONTAP 시스템에서 최대 전송 크기를 적절히 구성해야 합니다. 를 참조하세요"ONTAP 문서" 자세한 내용은.

   참고: RDMA를 통한 NFS를 사용하려면 ONTAP 시스템에서 RDMA를 통한 NFS를 구성해야 합니다. 를 참조하세요"ONTAP 문서" 자세한 내용은.

   참고: 다음 예에서는 StorageClass 정의 파일의 storagePool 필드에 특정 백엔드가 지정됩니다.

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

2. NetApp 또한 섹션에서 생성한 FlexVol 지원 Trident 백엔드에 해당하는 StorageClass를 생성할 것을 권장합니다 ."AIPod 배포를 위한 Trident 백엔드 예시" , 2단계. 다음 예제 명령은 FlexVol 볼륨에 대한 단일 StorageClass를 생성하는 방법을 보여줍니다.

   참고: 다음 예에서는 StorageClass 정의 파일의 storagePool 필드에 특정 백엔드가 지정되지 않았습니다. 이 StorageClass를 사용하여 Kubernetes를 사용하여 볼륨을 관리하는 경우 Trident 해당 StorageClass를 사용하는 사용 가능한 백엔드를 사용하려고 시도합니다. `ontap-nas` 운전사.

```
$ 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
```

아파치 에어플로우

Apache Airflow 배포

이 섹션에서는 Kubernetes 클러스터에 Airflow를 배포하기 위해 완료해야 하는 작업을 설명합니다.

Kubernetes 외의 플랫폼에도 Airflow를 배포하는 것이 가능합니다. Kubernetes 이외의 플랫폼에 Airflow를 배포하는 것은 이 솔루션의 범위를 벗어납니다.

필수 조건

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

1. 이미 작동하는 Kubernetes 클러스터가 있습니다.
2. Kubernetes 클러스터에 NetApp Trident 이미 설치하고 구성했습니다. Trident 에 대한 자세한 내용은 다음을 참조하세요."Trident 문서" .

Helm 설치

Airflow는 Kubernetes의 인기 있는 패키지 관리자인 Helm을 사용하여 배포됩니다. Airflow를 배포하기 전에 배포 점프 호스트에 Helm을 설치해야 합니다. 배포 점프 호스트에 Helm을 설치하려면 다음을 따르세요. "설치 지침" 공식 Helm 문서에서.

기본 Kubernetes StorageClass 설정

Airflow를 배포하기 전에 Kubernetes 클러스터 내에서 기본 StorageClass를 지정해야 합니다. Airflow 배포

프로세스는 기본 StorageClass를 사용하여 새로운 영구 볼륨을 프로비저닝하려고 시도합니다. StorageClass가 기본 StorageClass로 지정되지 않으면 배포가 실패합니다. 클러스터 내에서 기본 StorageClass를 지정하려면 다음 지침을 따르세요."Kubeflow 배포" 부분. 클러스터 내에서 이미 기본 StorageClass를 지정한 경우 이 단계를 건너뛸 수 있습니다.

Helm을 사용하여 공기 흐름 배포

Helm을 사용하여 Kubernetes 클러스터에 Airflow를 배포하려면 배포 점프 호스트에서 다음 작업을 수행하세요.

1. Helm을 사용하여 Airflow를 배포하려면 다음을 따르세요. "배치 지침" Artifact Hub의 공식 Airflow 차트입니다. 다음 예제 명령은 Helm을 사용하여 Airflow를 배포하는 방법을 보여줍니다. 값을 수정, 추가 및/또는 제거합니다. `custom- values.yaml` 환경과 원하는 구성에 따라 필요에 따라 파일을 추가하세요.

```
$ 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
```

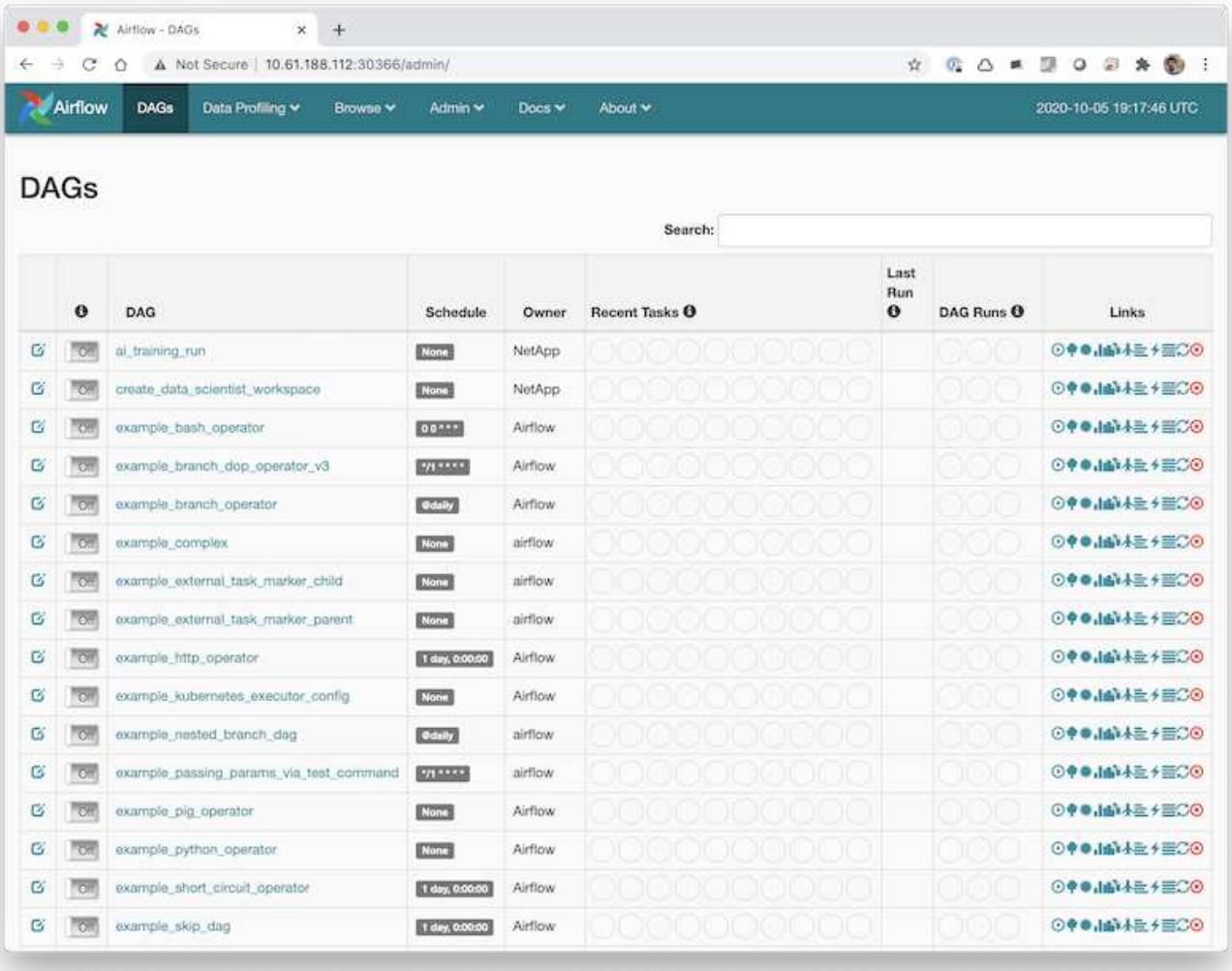
2. 모든 Airflow 포드가 제대로 작동하는지 확인하세요. 모든 포드가 시작되려면 몇 분이 걸릴 수 있습니다.

```
$ 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 Helm을 사용하여 Airflow를 배포할 때 콘솔에 인쇄된 지침에 따라 Airflow 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 확인하세요.



Airflow와 함께 NetApp DataOps Toolkit 사용

그만큼 "Kubernetes용 NetApp DataOps 툴킷" Airflow와 함께 사용할 수 있습니다. Airflow와 함께 NetApp DataOps Toolkit을 사용하면 스냅샷 및 복제본 생성과 같은 NetApp 데이터 관리 작업을 Airflow에서 조정하는 자동화된 워크플로에 통합할 수 있습니다.

를 참조하세요 "공기 흐름 예" Airflow와 함께 툴킷을 사용하는 방법에 대한 자세한 내용은 NetApp DataOps Toolkit GitHub 저장소 내 섹션을 참조하세요.

주피터허브

JupyterHub 배포

이 섹션에서는 Kubernetes 클러스터에 JupyterHub를 배포하기 위해 완료해야 하는 작업에 대해 설명합니다.

Kubernetes 외의 플랫폼에도 JupyterHub를 배포하는 것이 가능합니다. Kubernetes 이외의 플랫폼에 JupyterHub를 배포하는 것은 이 솔루션의 범위를 벗어납니다.

필수 조건

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

1. 이미 작동하는 Kubernetes 클러스터가 있습니다.
2. Kubernetes 클러스터에 NetApp Trident 이미 설치하고 구성했습니다. Trident 에 대한 자세한 내용은 다음을 참조하세요."Trident 문서" .

Helm 설치

JupyterHub는 Kubernetes의 인기 있는 패키지 관리자인 Helm을 사용하여 배포됩니다. JupyterHub를 배포하기 전에 Kubernetes 제어 노드에 Helm을 설치해야 합니다. Helm을 설치하려면 다음을 따르세요. "설치 지침" 공식 Helm 문서에서.

기본 **Kubernetes StorageClass** 설정

JupyterHub를 배포하기 전에 Kubernetes 클러스터 내에서 기본 StorageClass를 지정해야 합니다. 클러스터 내에서 기본 StorageClass를 지정하려면 다음 지침을 따르세요."Kubeflow 배포" 부분. 클러스터 내에서 기본 StorageClass를 이미 지정한 경우 이 단계를 건너뛸 수 있습니다.

JupyterHub 배포

위의 단계를 완료하면 이제 JupyterHub를 배포할 준비가 되었습니다. JupyterHub를 배포하려면 다음 단계가 필요합니다.

JupyterHub 배포 구성

배포하기 전에 각자의 환경에 맞게 JupyterHub 배포를 최적화하는 것이 좋습니다. **config.yaml** 파일을 만들고 Helm 차트를 사용하여 배포하는 동안 활용할 수 있습니다.

예제 **config.yaml** 파일은 다음에서 찾을 수 있습니다. https://github.com/jupyterhub/zero-to-jupyterhub-k8s/blob/HEAD/jupyterhub/values.yaml

이 config.yaml 파일에서 NetApp Trident StorageClass에 대한 **(singleuser.storage.dynamic.storageClass)** 매개변수를 설정할 수 있습니다. 이는 개별 사용자 작업 공간에 대한 볼륨을 프로비저닝하는 데 사용되는 스토리지 클래스입니다.

공유 볼륨 추가

모든 JupyterHub 사용자를 위한 공유 볼륨을 사용하려면 *config.yaml*을 적절히 조정하면 됩니다. 예를 들어, jupyterhub-shared-volume이라는 공유 PersistentVolumeClaim이 있는 경우 다음과 같이 모든 사용자 Pod에서 /home/shared로 마운트할 수 있습니다.

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

이는 선택 사항이며, 귀하의 필요에 맞게 이러한 매개변수를 조정할 수 있습니다.

Helm Chart로 JupyterHub 배포

Helm에 JupyterHub Helm 차트 저장소를 알립니다.

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

다음과 같은 출력이 표시됩니다.

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

이제 config.yaml이 포함된 디렉토리에서 다음 명령을 실행하여 config.yaml에 의해 구성된 차트를 설치하세요.

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

이 예에서는:

<helm-release-name>은 my-jupyterhub로 설정되며, 이는 JupyterHub 릴리스의 이름이 됩니다. <k8s-namespace>는 JupyterHub를 설치하려는 네임스페이스인 my-namespace로 설정됩니다. --create-namespace 플래그는 네임스페이스가 아직 존재하지 않을 경우 네임스페이스를 생성하는 데 사용됩니다. --values 플래그는 원하는 구성 옵션이 포함된 config.yaml 파일을 지정합니다.

배포 확인

2단계가 실행되는 동안 다음 명령을 실행하면 포드가 생성되는 것을 볼 수 있습니다.

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

허브와 프록시 포드가 실행 상태로 전환될 때까지 기다리세요.

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

JupyterHub에 접속하세요

JupyterHub에 접속하는 데 사용할 수 있는 IP를 찾으세요. 예제 출력과 같이 proxy-public 서비스의 EXTERNAL-IP를 사용할 수 있을 때까지 다음 명령을 실행합니다.

config.yaml 파일에서 NodePort 서비스를 사용했는데, 설정(예: LoadBalancer)에 따라 환경에 맞게 조정할 수 있습니다.

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

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

JupyterHub를 사용하려면 프록시-공개 서비스의 외부 IP를 브라우저에 입력하세요.

JupyterHub와 함께 NetApp DataOps Toolkit 사용

그만큼 "Kubernetes용 NetApp DataOps 툴킷" JupyterHub와 함께 사용할 수 있습니다. JupyterHub와 함께 NetApp DataOps Toolkit을 사용하면 최종 사용자는 Jupyter Notebook 내에서 직접 작업 공간 백업 및/또는 데이터 세트-모델 추적을 위한 볼륨 스냅샷을 만들 수 있습니다.

초기 설정

JupyterHub에서 DataOps Toolkit을 사용하려면 먼저 JupyterHub가 개별 사용자 Jupyter Notebook Server 포드에 할당한 Kubernetes 서비스 계정에 적절한 권한을 부여해야 합니다. JupyterHub는 다음에 의해 지정된 서비스 계정을 사용합니다. `singleuser.serviceAccountName` JupyterHub Helm 차트 구성 파일의 변수입니다.

DataOps Toolkit에 대한 클러스터 역할 생성

먼저, 볼륨 스냅샷을 만드는 데 필요한 Kubernetes API 권한이 있는 'netapp-dataops'라는 클러스터 역할을 만듭니다.

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

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

Notebook Server 서비스 계정에 클러스터 역할 할당

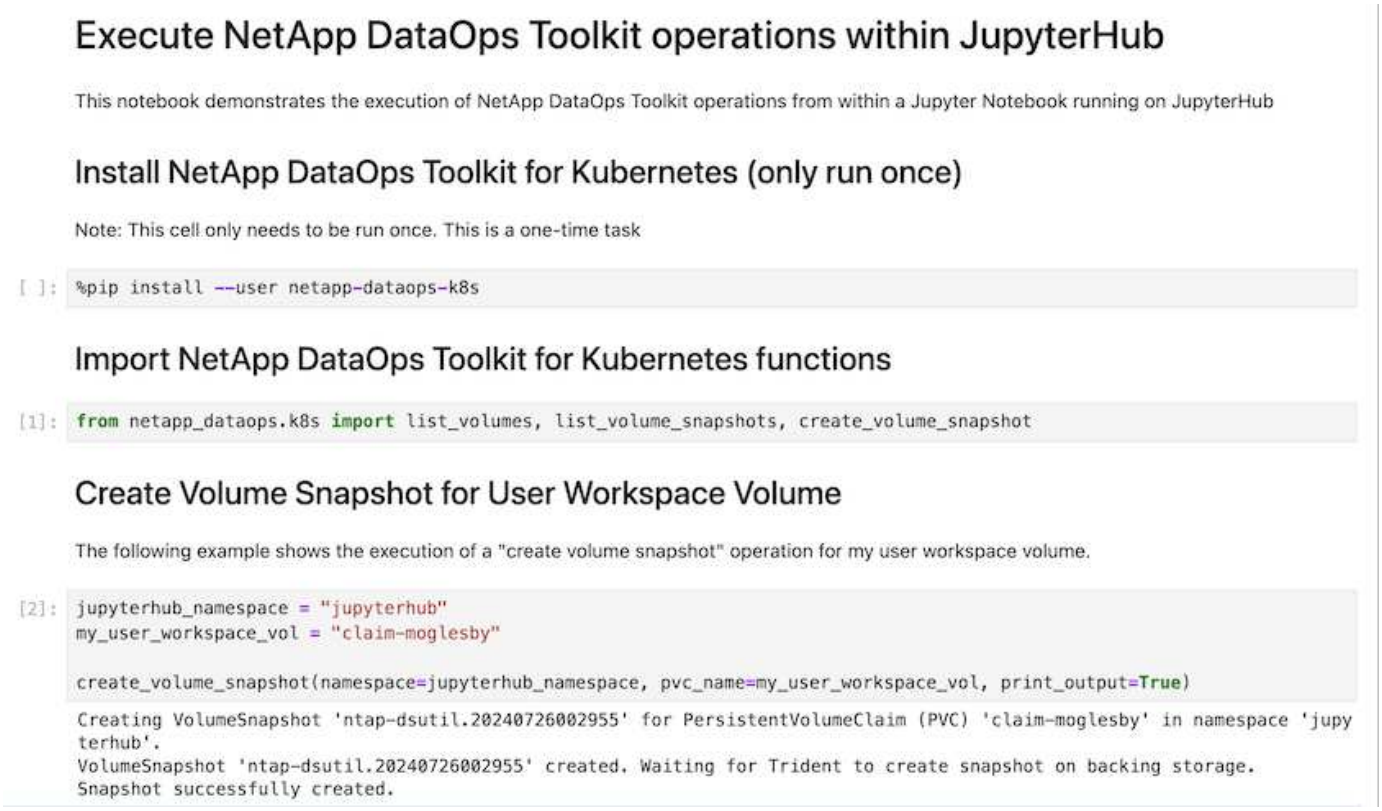
적절한 네임스페이스의 적절한 서비스 계정에 'netapp-dataops-snapshots' 클러스터 역할을 할당하는 역할 바인딩을 만듭니다. 예를 들어, 'jupyterhub' 네임스페이스에 JupyterHub를 설치하고 다음을 통해 '기본' 서비스 계정을 지정한 경우 `singleuser.serviceAccountName` 다음 예에서 보듯이 변수를 사용하면 'jupyterhub' 네임스페이스의 '기본' 서비스 계정에 'netapp-dataops-snapshots' 클러스터 역할을 할당할 수 있습니다.

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

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

Jupyter Notebook 내에서 볼륨 스냅샷 만들기

이제 JupyterHub 사용자는 다음 예에서 볼 수 있듯이 NetApp DataOps Toolkit을 사용하여 Jupyter Notebook 내에서 직접 볼륨 스냅샷을 만들 수 있습니다.



NetApp SnapMirror 사용하여 JupyterHub에 데이터 수집

NetApp SnapMirror 는 NetApp 스토리지 시스템 간에 데이터를 복제할 수 있는 복제 기술입니다. SnapMirror 사용하면 원격 환경에서 JupyterHub로 데이터를 수집할 수 있습니다.

예제 워크플로 및 데모

참조하다"이 Tech ONTAP 블로그 게시물" NetApp SnapMirror 사용하여 JupyterHub에 데이터를 수집하는 방법에 대한 자세한 예제 워크플로와 데모를 확인하세요.

ML플로우

MLflow 배포

이 섹션에서는 Kubernetes 클러스터에 MLflow를 배포하기 위해 완료해야 하는 작업에 대해 설명합니다.

Kubernetes 외의 플랫폼에도 MLflow를 배포하는 것이 가능합니다. Kubernetes 이외의 플랫폼에 MLflow를 배포하는 것은 이 솔루션의 범위를 벗어납니다.

필수 조건

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

1. 이미 작동하는 Kubernetes 클러스터가 있습니다.

2. Kubernetes 클러스터에 NetApp Trident 이미 설치하고 구성했습니다. Trident 에 대한 자세한 내용은 다음을 참조하세요."Trident 문서" .

Helm 설치

MLflow는 Kubernetes의 인기 있는 패키지 관리자인 Helm을 사용하여 배포됩니다. MLflow를 배포하기 전에 Kubernetes 제어 노드에 Helm을 설치해야 합니다. Helm을 설치하려면 다음을 따르세요. "설치 지침" 공식 Helm 문서에서.

기본 Kubernetes StorageClass 설정

MLflow를 배포하기 전에 Kubernetes 클러스터 내에서 기본 StorageClass를 지정해야 합니다. 클러스터 내에서 기본 StorageClass를 지정하려면 다음 지침을 따르세요."Kubeflow 배포" 부분. 클러스터 내에서 기본 StorageClass를 이미 지정한 경우 이 단계를 건너뛸 수 있습니다.

MLflow 배포

필수 조건을 충족하면 Helm 차트를 사용하여 MLflow 배포를 시작할 수 있습니다.

MLflow Helm 차트 배포를 구성합니다.

Helm 차트를 사용하여 MLflow를 배포하기 전에 **config.yaml** 파일을 사용하여 NetApp Trident Storage Class를 사용하도록 배포를 구성하고 다른 매개변수를 요구 사항에 맞게 변경할 수 있습니다. **config.yaml** 파일의 예는 다음에서 찾을 수 있습니다. https://github.com/bitnami/charts/blob/main/bitnami/mlflow/values.yaml

config.yaml 파일의 **global.defaultStorageClass** 매개변수에서 Trident storageClass를 설정할 수 있습니다(예: storageClass: "ontap-flexvol").

Helm 차트 설치

다음 명령을 사용하여 MLflow용 사용자 정의 **config.yaml** 파일로 Helm 차트를 설치할 수 있습니다.

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

이 명령은 제공된 **config.yaml** 파일을 통해 사용자 정의 구성으로 Kubernetes 클러스터에 MLflow를 배포합니다. MLflow는 지정된 네임스페이스에 배포되고, 릴리스에 대한 무작위 릴리스 이름은 Kubernetes를 통해 제공됩니다.

배포 확인

Helm 차트 배포가 완료되면 다음을 사용하여 서비스에 액세스할 수 있는지 확인할 수 있습니다.

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

*jupyterhub*를 배포 중에 사용한 네임스페이스로 바꾸세요.

다음 서비스가 표시되어야 합니다.

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

MLflow에 포트 30002로 접근하기 위해 NodePort 서비스를 사용하도록 config.yaml 파일을 편집했습니다.

MLflow에 접속하세요

MLflow와 관련된 모든 서비스가 실행되면 지정된 NodePort 또는 LoadBalancer IP 주소(예: `http://10.61.181.109:30002`)

NetApp 및 MLflow를 사용한 데이터 세트-모델 추적성

그만큼 "Kubernetes용 NetApp DataOps 툴킷" MLflow의 실험 추적 기능과 함께 사용하여 데이터 세트-모델 또는 작업 공간-모델 추적을 구현할 수 있습니다.

데이터세트-모델 또는 작업 공간-모델 추적을 구현하려면 다음 예제 코드 조각과 같이 교육 실행의 일부로 DataOps Toolkit을 사용하여 데이터세트 또는 작업 공간 볼륨의 스냅샷을 만들기만 하면 됩니다. 이 코드는 MLflow 실험 추적 서버에 로깅하는 특정 교육 실행과 연관된 태그로 데이터 볼륨 이름과 스냅샷 이름을 저장합니다.

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

with mlflow.start_run() :
    ...

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

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

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

    ...
```

쿠베플로우

Kubeflow 배포

이 섹션에서는 Kubernetes 클러스터에 Kubeflow를 배포하기 위해 완료해야 하는 작업을 설명합니다.

필수 조건

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

1. 이미 작동하는 Kubernetes 클러스터가 있으며, 배포하려는 Kubeflow 버전에서 지원하는 Kubernetes 버전을 실행하고 있습니다. 지원되는 Kubernetes 버전 목록은 Kubeflow 버전에 대한 종속성을 참조하세요."공식 Kubeflow 문서" .
2. Kubernetes 클러스터에 NetApp Trident 이미 설치하고 구성했습니다. Trident 에 대한 자세한 내용은 다음을 참조하세요."Trident 문서" .

기본 **Kubernetes StorageClass** 설정

Kubeflow를 배포하기 전에 Kubernetes 클러스터 내에서 기본 StorageClass를 지정하는 것이 좋습니다. Kubeflow 배포 프로세스는 기본 StorageClass를 사용하여 새로운 영구 볼륨을 프로비저닝하려고 시도할 수 있습니다. StorageClass가 기본 StorageClass로 지정되지 않으면 배포가 실패할 수 있습니다. 클러스터 내에서 기본 StorageClass를 지정하려면 배포 점프 호스트에서 다음 작업을 수행하세요. 클러스터 내에서 기본 StorageClass를 이미 지정한 경우 이 단계를 건너뛸 수 있습니다.

1. 기존 StorageClass 중 하나를 기본 StorageClass로 지정합니다. 다음 예제 명령은 StorageClass라는 이름의 지정을 보여줍니다. `ontap-ai-flexvols-retain` 기본 StorageClass로.

> 그만큼 `ontap-nas-flexgroup` Trident 백엔드 유형은 최소 PVC 크기가 상당히 큽니다. 기본적으로 Kubeflow는 크기가 몇 GB에 불과한 PVC를 프로비저닝하려고 시도합니다. 따라서 StorageClass를 사용하는 StorageClass를 지정해서는 안 됩니다. `ontap-nas-flexgroup` Kubeflow 배포를 위해 기본 StorageClass로 백엔드 유형을 지정합니다.

```
$ 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
```

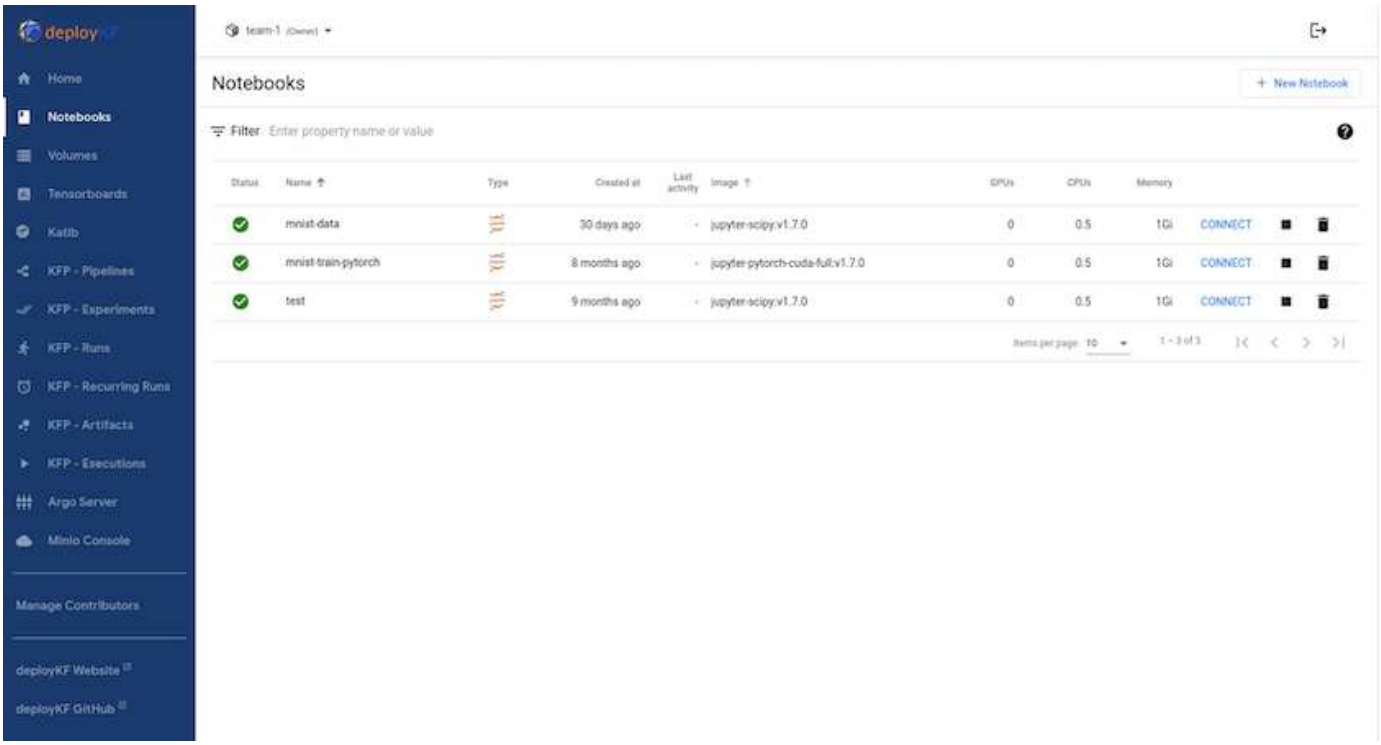
Kubeflow 배포 옵션

Kubeflow를 배포하는 데에는 다양한 옵션이 있습니다. 를 참조하세요"공식 Kubeflow 문서" 배포 옵션 목록을 확인하고, 귀하의 요구 사항에 가장 적합한 옵션을 선택하세요.

> 검증 목적으로 Kubeflow 1.7을 배포했습니다."배포KF" 0.1.1.

데이터 과학자 또는 개발자를 위한 **Jupyter Notebook** 작업 공간 프로비저닝

Kubeflow는 데이터 과학자의 작업 공간 역할을 하는 새로운 Jupyter Notebook 서버를 빠르게 프로비저닝할 수 있습니다. Kubeflow 컨텍스트 내 Jupyter Notebook에 대한 자세한 내용은 다음을 참조하세요. "공식 Kubeflow 문서" .



Kubeflow와 함께 NetApp DataOps Toolkit 사용

그만큼 "Kubernetes를 위한 NetApp 데이터 과학 툴킷" Kubeflow와 함께 사용할 수 있습니다. Kubeflow와 함께 NetApp 데이터 과학 툴킷을 사용하면 다음과 같은 이점이 있습니다.

- 데이터 과학자는 Jupyter Notebook 내에서 직접 스냅샷 및 복제본 생성과 같은 고급 NetApp 데이터 관리 작업을 수행할 수 있습니다.
- Kubeflow Pipelines 프레임워크를 사용하면 스냅샷 및 복제본 생성과 같은 고급 NetApp 데이터 관리 작업을 자동화된 워크플로에 통합할 수 있습니다.

를 참조하세요 "Kubeflow 예제" Kubeflow와 함께 툴킷을 사용하는 방법에 대한 자세한 내용은 NetApp Data Science Toolkit GitHub 저장소 내 섹션을 참조하세요.

예제 워크플로 - Kubeflow와 NetApp DataOps 툴킷을 사용하여 이미지 인식 모델 학습

이 섹션에서는 Kubeflow와 NetApp DataOps Toolkit을 사용하여 이미지 인식을 위한 신경망을 훈련하고 배포하는 데 필요한 단계를 설명합니다. 이는 NetApp 스토리지를 통합한 교육 작업을 보여주는 예시로 사용됩니다.

필수 조건

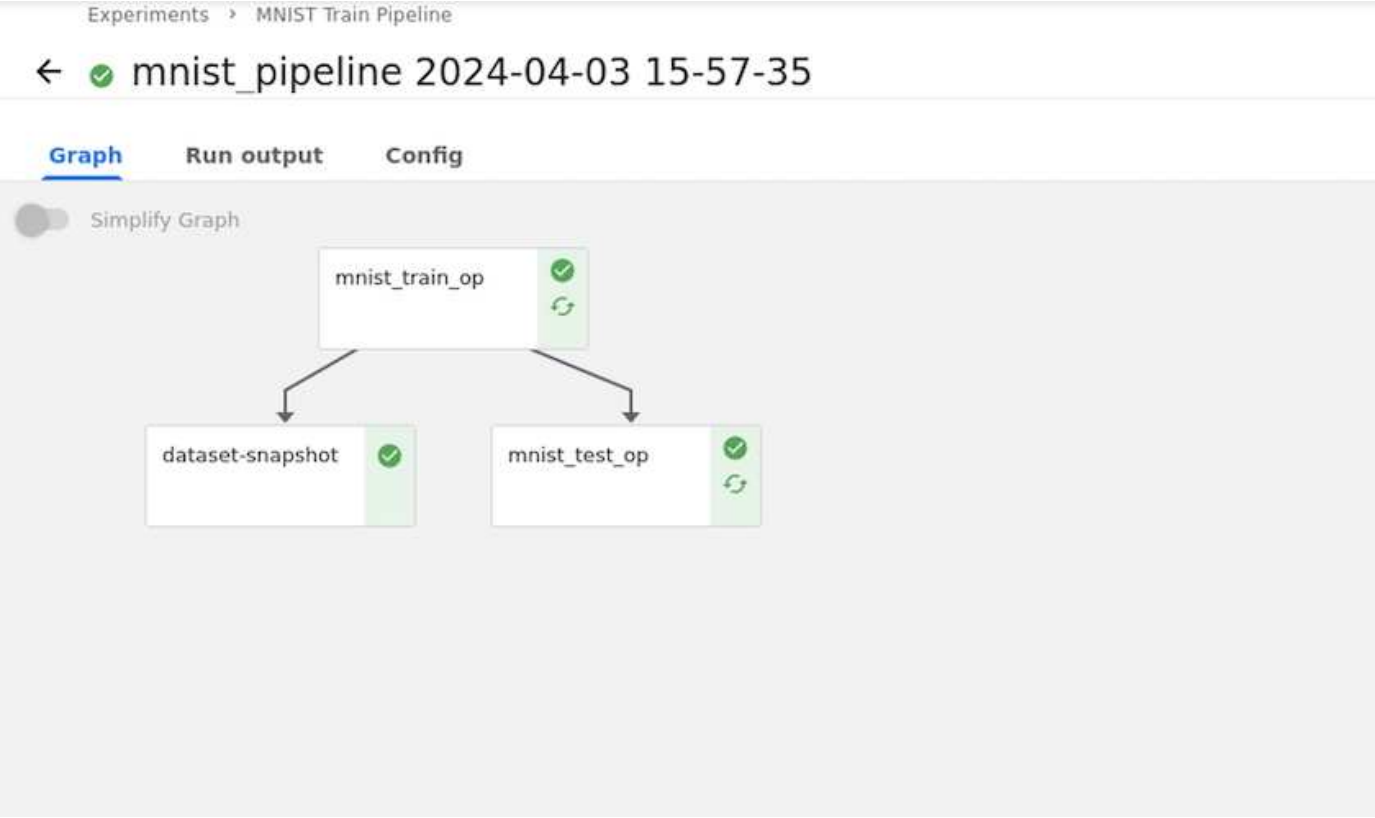
Kubeflow 파이프라인 내의 학습 및 테스트 단계에 사용할 필수 구성을 포함하는 Dockerfile을 만듭니다. 다음은 Dockerfile의 예입니다.

```
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 라이브러리와 패키지를 설치하세요. 머신 러닝 모델을 학습하기 전에 이미 작동하는 Kubeflow 배포가 있다고 가정합니다.

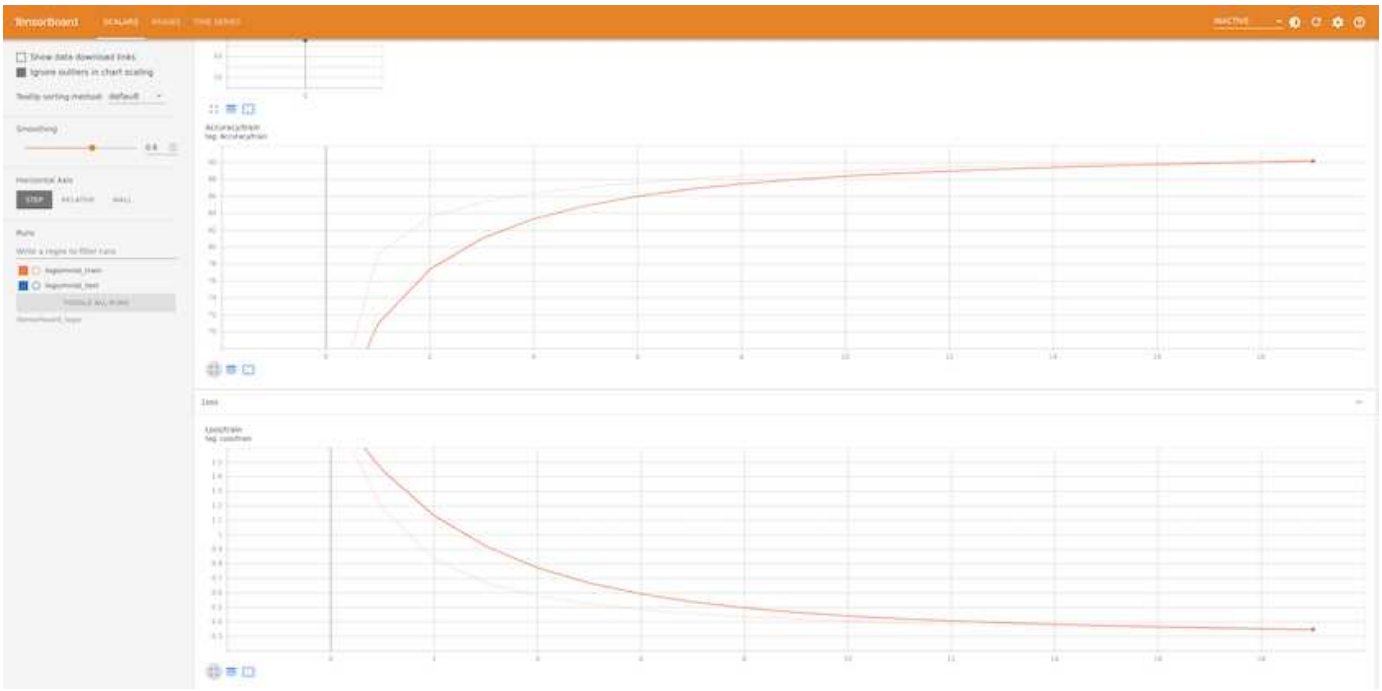
PyTorch와 **Kubeflow** 파이프라인을 사용하여 **MNIST** 데이터에서 소규모 **NN** 학습

우리는 MNIST 데이터로 훈련된 작은 신경망의 예를 사용합니다. MNIST 데이터 세트는 0~9까지의 숫자로 쓰인 손으로 쓴 이미지로 구성되어 있습니다. 이미지 크기는 28x28픽셀입니다. 데이터 세트는 60,000개의 훈련 이미지와 10,000개의 검증 이미지로 나뉩니다. 이 실험에 사용된 신경망은 2계층 피드포워드 네트워크입니다. Kubeflow Pipelines를 사용하여 훈련이 실행됩니다. 문서를 참조하세요 "여기" 자세한 내용은. Kubeflow 파이프라인은 필수 구성 요소 섹션의 Docker 이미지를 통합합니다.



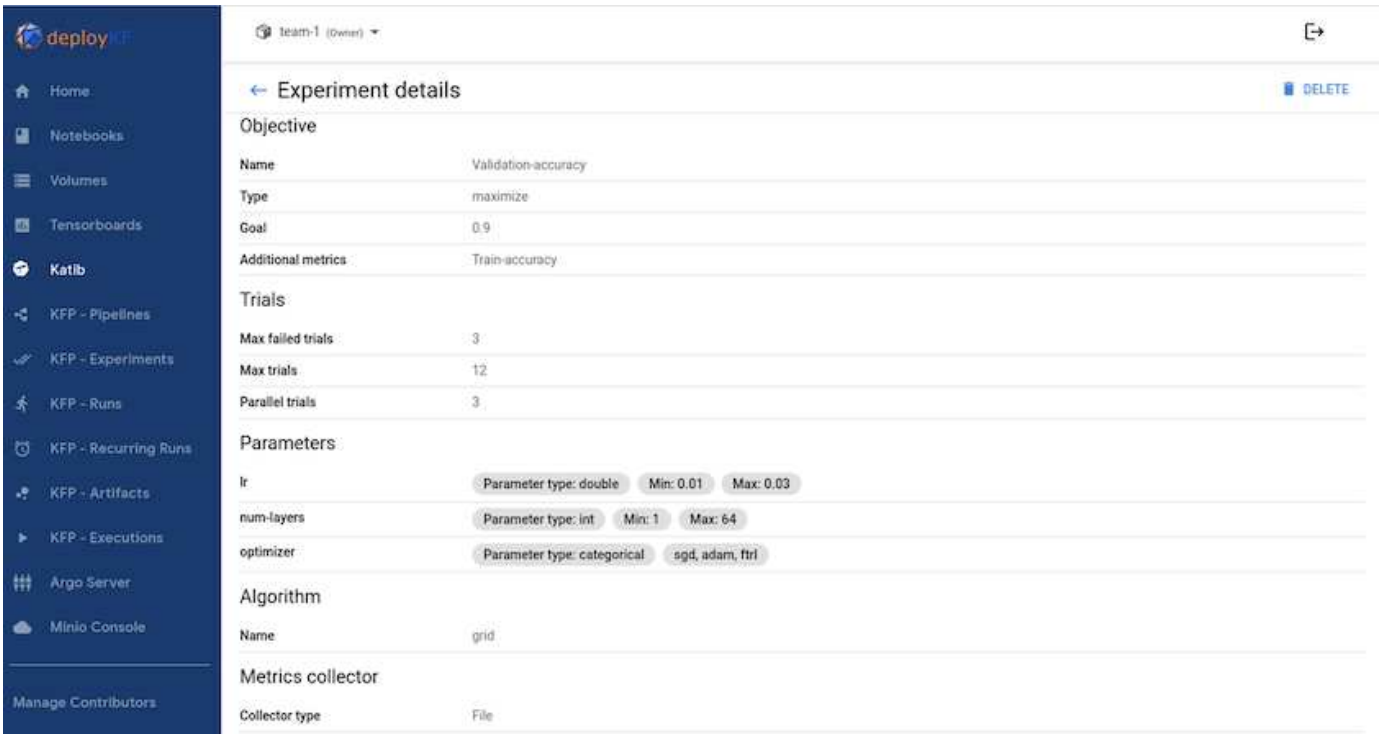
Tensorboard를 사용하여 결과 시각화

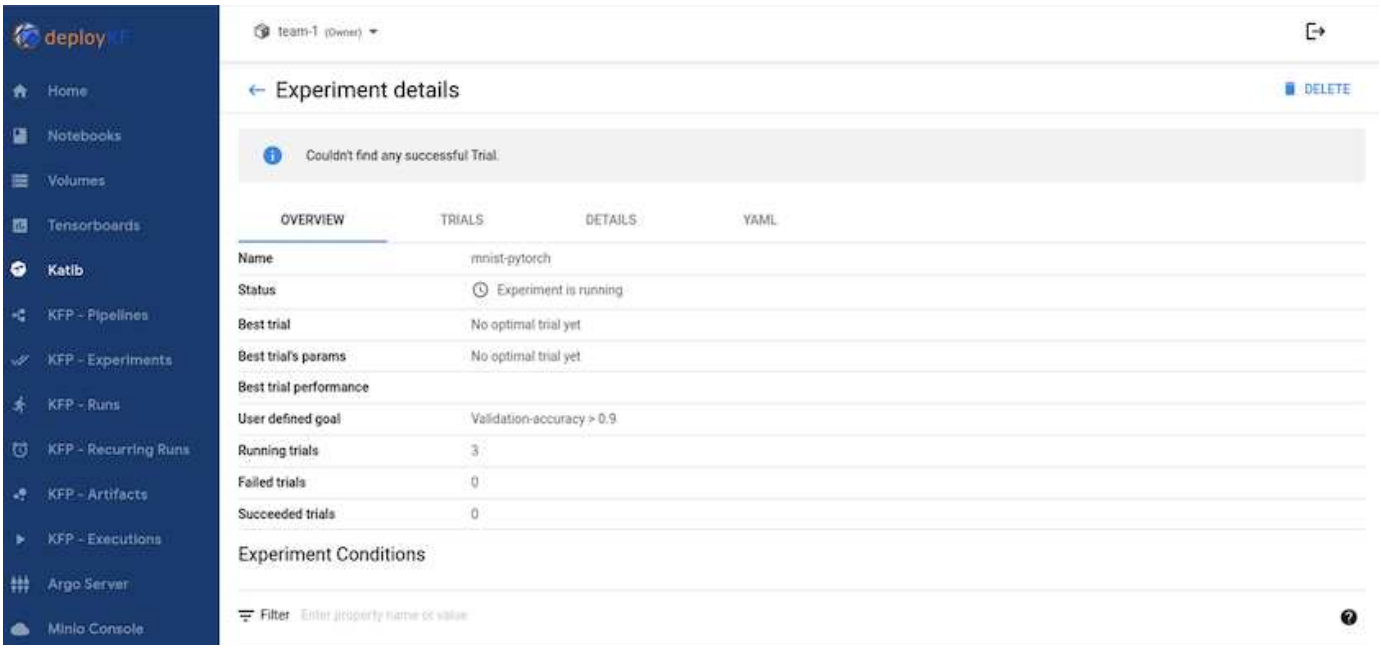
모델이 훈련되면 Tensorboard를 사용하여 결과를 시각화할 수 있습니다. "텐서보드" Kubeflow 대시보드의 기능으로 사용할 수 있습니다. 귀하의 작업에 맞는 사용자 정의 텐서보드를 만들 수 있습니다. 아래 예는 학습 정확도 대 에포크 수, 학습 손실 대 에포크 수의 플롯을 보여줍니다.



Katib을 사용하여 하이퍼파라미터 실험

"카티브"Kubeflow 내의 도구로, 모델 하이퍼파라미터를 실험하는 데 사용할 수 있습니다. 실험을 만들려면 먼저 원하는 지표/목표를 정의하세요. 이는 일반적으로 테스트 정확도입니다. 지표가 정의되면, 조정하고 싶은 하이퍼 매개변수(최적화 도구/학습 속도/계층 수)를 선택합니다. Katib은 사용자 정의 값으로 하이퍼파라미터 스윕을 수행하여 원하는 지표를 만족하는 최적의 매개변수 조합을 찾습니다. UI의 각 섹션에서 이러한 매개변수를 정의할 수 있습니다. 또는 필요한 사양을 담은 **YAML** 파일을 정의할 수 있습니다. 아래는 Katib 실험의 그림입니다.





NetApp 스냅샷을 사용하여 추적을 위한 데이터 저장

모델 학습 중에 추적을 위해 학습 데이터 세트의 스냅샷을 저장하고 싶을 수도 있습니다. 이를 위해 아래와 같이 파이프라인에 스냅샷 단계를 추가할 수 있습니다. 스냅샷을 생성하려면 다음을 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)
```

를 참조하세요 "Kubeflow를 위한 NetApp DataOps Toolkit 예제" 자세한 내용은.

Trident 작업 예시

이 섹션에는 Trident 사용하여 수행할 수 있는 다양한 작업의 예가 포함되어 있습니다.

기존 볼륨 가져오기

Kubernetes 클러스터 내의 컨테이너에 마운트하려는 NetApp 스토리지 시스템/플랫폼에 기존 볼륨이 있지만 해당 볼륨이 클러스터의 PVC에 연결되지 않은 경우 해당 볼륨을 가져와야 합니다. Trident 볼륨 가져오기 기능을 사용하면 이러한 볼륨을 가져올 수 있습니다.

다음 예제 명령은 이름이 지정된 볼륨을 가져오는 방법을 보여줍니다. `pb_fg_all` . PVC에 대한 자세한 내용은 다음을 참조하세요. "공식 Kubernetes 문서" . 볼륨 가져오기 기능에 대한 자세한 내용은 다음을 참조하세요. "Trident 문서" .

안 `accessModes` 의 가치 `ReadOnlyMany` 예제 PVC 사양 파일에 지정되어 있습니다. 자세한 내용은 `accessMode` 필드에서 확인하세요 "공식 Kubernetes 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 볼륨 제공

Trident 사용하면 NetApp 스토리지 시스템이나 플랫폼에 새 볼륨을 프로비저닝할 수 있습니다.

kubectl을 사용하여 새 볼륨 프로비저닝

다음 예제 명령은 kubectl을 사용하여 새로운 FlexVol volume 프로비저닝하는 방법을 보여줍니다.

안 `accessModes` 의 가치 `ReadWriteMany` 다음 예제 PVC 정의 파일에 지정되어 있습니다. 자세한 내용은 `accessMode` 필드에서 확인하세요 "공식 Kubernetes 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 볼륨 프로비저닝

Kubernetes용 NetApp DataOps Toolkit을 사용하여 NetApp 스토리지 시스템이나 플랫폼에 새 볼륨을 프로비저닝할 수도 있습니다. Kubernetes용 NetApp DataOps Toolkit은 Trident 사용하여 볼륨을 프로비저닝하지만 사용자를 위해 프로세스를 간소화합니다. 를 참조하세요"선적 서류 비치" 자세한 내용은.

AIPod 배포를 위한 고성능 작업 예시

단일 노드 AI 워크로드 실행

Kubernetes 클러스터에서 단일 노드 AI 및 ML 작업을 실행하려면 배포 점프 호스트에서 다음 작업을 수행하세요. Trident 사용하면 페타바이트 규모의 데이터를 포함하는 데이터 볼륨을 Kubernetes 워크로드에서 쉽고 빠르게 액세스할 수 있습니다. Kubernetes 포드 내에서 이러한 데이터 볼륨에 액세스할 수 있게 하려면 포드 정의에서 PVC를 지정하기만 하면 됩니다.

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

1. 다음 예제 명령은 ImageNet 데이터 세트를 사용하는 TensorFlow 벤치마크 워크로드에 대한 Kubernetes 작업을 생성하는 방법을 보여줍니다. ImageNet 데이터 세트에 대한 자세한 내용은 다음을 참조하세요. "ImageNet 웹사이트" .

   이 예제 작업은 8개의 GPU를 요청하므로 8개 이상의 GPU를 갖춘 단일 GPU 워커 노드에서 실행될 수 있습니다. 이 예제 작업은 8개 이상의 GPU를 갖춘 작업자 노드가 없거나 현재 다른 작업 부하로 인해 사용 중인 클러스터에 제출될 수 있습니다. 그렇다면 해당 작업은 해당 작업자 노드가 사용 가능해질 때까지 보류 상태로 유지됩니다.

   또한, 저장 대역폭을 최대화하기 위해 이 작업이 생성하는 포드 내에 필요한 학습 데이터가 포함된 볼륨이 두 번 마운트됩니다. 또 다른 볼륨도 포드에 마운트됩니다. 두 번째 볼륨은 결과와 측정 항목을 저장하는 데 사용됩니다. 이러한 볼륨은 PVC의 이름을 사용하여 작업 정의에 참조됩니다. Kubernetes 작업에 대한 자세한 내용은 다음을 참조하세요. "공식 Kubernetes 문서" .

   안 `emptyDir` 볼륨이 있는 `medium` 의 가치 `Memory` 에 마운트됩니다 `/dev/shm` 이 예제 작업이 생성하는 포드에서. 기본 크기 `/dev/shm` Docker 컨테이너 런타임에서 자동으로 생성되는 가상 볼륨은 때때로 TensorFlow의 요구 사항을 충족하지 못할 수 있습니다. 장착 `emptyDir` 다음 예와 같이 볼륨은 충분히 큰 크기를 제공합니다. `/dev/shm` 가상 볼륨. 더 많은 정보를 원하시면 `emptyDir` 볼륨을 참조하십시오 "공식 Kubernetes 문서" .

   이 예제 작업 정의에 지정된 단일 컨테이너에는 다음이 제공됩니다. `securityContext > privileged` 의 가치 `true` . 이 값은 컨테이너가 호스트에 대한 루트 액세스 권한을 효과적으로 가지고 있음을 의미합니다. 이 주석은 실행되는 특정 작업 부하에 루트 액세스가 필요하기 때문에 이 경우에 사용됩니다. 구체적으로, 워크로드가 수행하는 캐시 지우기 작업에는 루트 액세스가 필요합니다. 이것이든 아니든 `privileged: true` 주석이 필요한지는 실행 중인 특정 작업 부하의 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 생성한 작업이 올바르게 실행되는지 확인하세요. 다음 예제 명령은 작업 정의에 지정된 대로 작업에 대한 단일 포드가 생성되었으며, 이 포드가 현재 GPU 워커 노드 중 하나에서 실행 중인지 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 완료되는지 확인하세요. 다음 예제 명령은 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 아티팩트를 정리합니다. 다음 예제 명령은 1단계에서 생성된 작업 객체를 삭제하는 방법을 보여줍니다.

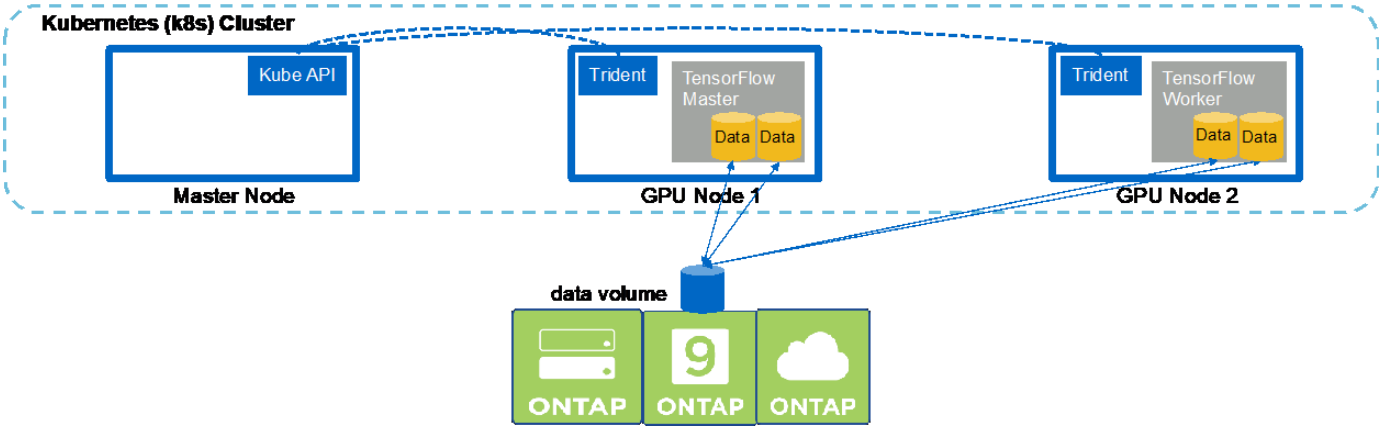
   작업 객체를 삭제하면 Kubernetes는 연관된 모든 Pod를 자동으로 삭제합니다.

```
$ 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 멀티노드 AI 및 ML 작업을 실행하려면 배포 점프 호스트에서 다음 작업을 수행하세요. 이 프로세스를 사용하면 NetApp 볼륨에 저장된 데이터를 활용하고 단일 작업자 노드가 제공할 수 있는 것보다 많은 GPU를 사용할 수 있습니다. 동기식 분산 AI 작업을 묘사한 다음 그림을 참조하세요.

ⓘ 동기 분산 작업은 비동기 분산 작업에 비해 성능과 교육 정확도를 높이는 데 도움이 될 수 있습니다. 동기 작업과 비동기 작업의 장단점에 대한 논의는 이 문서의 범위를 벗어납니다.



1. 다음 예제 명령은 섹션의 예제에서 단일 노드에서 실행된 동일한 TensorFlow 벤치마크 작업의 동기 분산 실행에 참여하는 하나의 작업자 생성을 보여줍니다."단일 노드 AI 워크로드 실행" . 이 특정 예에서 작업은 두 개의 작업자 노드에서 실행되므로 단일 작업자만 배포됩니다.

   이 예제 워커 배포에서는 8개의 GPU를 요청하므로 8개 이상의 GPU가 있는 단일 GPU 워커 노드에서 실행될 수 있습니다. GPU 워커 노드에 8개 이상의 GPU가 있는 경우 성능을 극대화하려면 워커 노드가 제공하는 GPU 수와 같도록 이 숫자를 늘리는 것이 좋습니다. Kubernetes 배포에 대한 자세한 내용은 다음을 참조하세요. "공식

Kubernetes 문서" .

이 예에서는 특정 컨테이너화된 작업자가 스스로 완료될 수 없기 때문에 Kubernetes 배포가 생성됩니다. 따라서 Kubernetes 작업 구조를 사용하여 배포하는 것은 의미가 없습니다. 작업자가 자체적으로 완료되도록 설계되거나 작성된 경우, 작업 구성을 사용하여 작업자를 배치하는 것이 합리적일 수 있습니다.

이 예제 배포 사양에 지정된 포드는 다음과 같습니다. `hostNetwork` 의 가치 `true` . 이 값은 Kubernetes가 일반적으로 각 Pod에 대해 생성하는 가상 네트워킹 스택 대신, Pod가 호스트 워커 노드의 네트워킹 스택을 사용한다는 것을 의미합니다. 이 주석이 사용되는 이유는 특정 워크로드가 동기식 분산 방식으로 워크로드를 실행하기 위해 Open MPI, NCCL 및 Horovod에 의존하기 때문입니다. 따라서 호스트 네트워킹 스택에 액세스해야 합니다. Open MPI, NCCL, Horovod에 대한 논의는 이 문서의 범위를 벗어납니다. 이것이든 아니든 `hostNetwork: true` 주석이 필요한지는 실행 중인 특정 작업 부하의 요구 사항에 따라 달라집니다. 자세한 내용은 `hostNetwork` 필드에서 확인하세요 "공식 Kubernetes 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 생성한 작업자 배포가 성공적으로 시작되었는지 확인하세요. 다음 예제 명령은 배포 정의에 표시된 대로 배포에 대해 단일 워커 포드가 생성되었으며, 이 포드가 현재 GPU 워커 노드 중 하나에서 실행 중인지 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. 동기식 멀티노드 작업을 시작하고, 참여하고, 실행을 추적하는 마스터에 대한 Kubernetes 작업을 만듭니다. 다음 예제 명령은 예제 섹션의 단일 노드에서 실행된 동일한 TensorFlow 벤치마크 작업의 동기 분산 실행을 시작하고 참여하고 추적하는 하나의 마스터를 생성합니다."단일 노드 AI 워크로드 실행" .

   이 예제 마스터 작업은 8개의 GPU를 요청하므로 8개 이상의 GPU가 있는 단일 GPU 워커 노드에서 실행될 수 있습니다. GPU 워커 노드에 8개 이상의 GPU가 있는 경우 성능을 극대화하려면 워커 노드가 제공하는 GPU 수와 같도록 이 숫자를 늘리는 것이 좋습니다.

   이 예제 작업 정의에 지정된 마스터 포드에는 다음이 제공됩니다. `hostNetwork` 의 가치 `true` , 작업자 포드에 다음과 같은 것이 주어진 것처럼 `hostNetwork` 의 가치 `true` 1단계에서. 이 값이 필요한 이유에 대한 자세한

내용은 1단계를 참조하세요.

```
$ 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 생성한 마스터 작업이 올바르게 실행되는지 확인하세요. 다음 예제 명령은 작업 정의에 표시된 대로 작업에 대한 단일 마스터 포드가 생성되었으며, 이 포드가 현재 GPU 워커 노드 중 하나에서 실행 중인지 확인합니다. 1단계에서 처음 본 워커 포드가 여전히 실행 중인지, 마스터 포드와 워커 포드가 다른 노드에서 실행 중인지도 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 생성한 마스터 작업이 성공적으로 완료되는지 확인하세요. 다음 예제 명령은 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. 더 이상 필요하지 않으면 작업자 배포를 삭제하세요. 다음 예제 명령은 1단계에서 생성된 작업자 배포 개체를 삭제하는 방법을 보여줍니다.

   워커 배포 객체를 삭제하면 쿠버네티스는 연관된 모든 워커 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 정리합니다. 다음 예제 명령은 3단계에서 생성된 마스터 작업 개체를 삭제하는 방법을 보여줍니다.

   마스터 작업 객체를 삭제하면 Kubernetes는 연관된 모든 마스터 포드를 자동으로 삭제합니다.

```
$ 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.
```