NetApp的開源 MLOps

NetApp artificial intelligence solutions

NetApp
August 18, 2025

目錄

NetApp的開源 MLOps

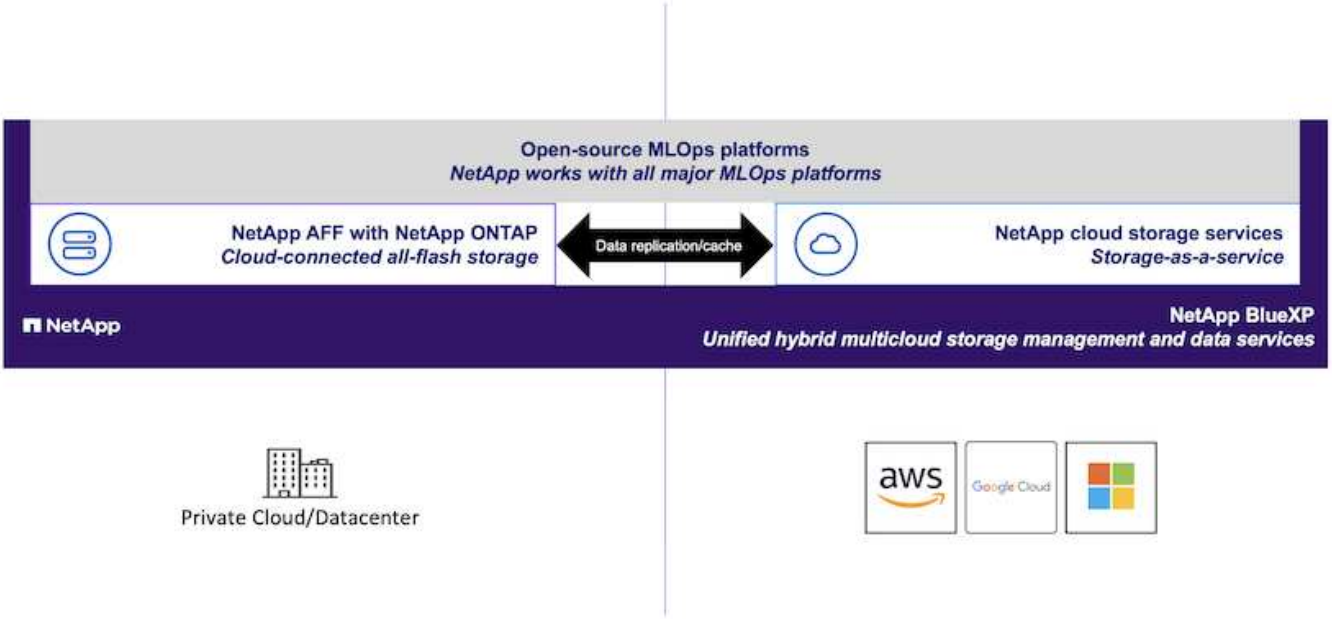
NetApp的開源 MLOps

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

各種規模和各行業的公司和組織都在轉向人工智慧 (AI) 來解決現實世界的問題、提供創新的產品和服務，並在競爭日益激烈的市場中佔據優勢。許多組織正在轉向開源 MLOps 工具，以跟上產業快速創新的步伐。這些開源工具提供了先進的功能和尖端的特性，但通常不考慮資料可用性和資料安全性。不幸的是，這意味著高技能的資料科學家被迫花費大量時間等待獲取資料或等待基本的資料相關操作完成。透過將流行的開源 MLOps 工具與NetApp的智慧資料基礎架構結合，組織可以加速其資料管道，從而加速其 AI 計劃。他們可以從資料中釋放價值，同時確保資料受到保護且安全。該解決方案展示了NetApp資料管理功能與幾種流行的開源工具和框架的配對，以應對這些挑戰。

以下列表重點介紹了此解決方案支援的一些關鍵功能：

- 使用者可以快速配置由高效能、橫向擴展的NetApp儲存支援的新的高容量資料磁碟區和開發工作區。
- 使用者可以幾乎即時地克隆大容量資料磁碟區和開發工作區，以便進行實驗或快速迭代。
- 使用者可以幾乎即時保存大容量資料磁碟區和開發工作區的快照，以進行備份和/或可追溯性/基準測試。



典型的 MLOps 工作流程包含開發工作區，通常採用以下形式"Jupyter 筆記本"；實驗追蹤；自動化訓練管道；資料管道；以及推理/部署。該解決方案重點介紹了幾種不同的工具和框架，它們可以獨立使用或結合使用來解決工作流程的不同方面。我們也展示了NetApp資料管理功能與每種工具的配對。此解決方案旨在提供建置模組，組織可據此建置針對其用例和要求的客製化 MLOps 工作流程。

此解決方案涵蓋以下工具/框架：

- "Apache Airflow"

- "JupyterHub"
- "Kubeflow"
- "機器學習流"

以下列表描述了獨立或共同部署這些工具的常見模式。

- 聯合部署 JupyterHub、MLflow 和 Apache Airflow - JupyterHub"Jupyter 筆記本" 、用於實驗追蹤的 MLflow 以及用於自動化訓練和資料管道的 Apache Airflow。
- 聯合部署 Kubeflow 和 Apache Airflow - Kubeflow"Jupyter 筆記本" 、實驗追蹤、自動化訓練管道和推理；以及用於資料管道的 Apache Airflow。
- 將 Kubeflow 部署為一體化 MLOps 平台解決方案"Jupyter 筆記本"、實驗追蹤、自動化訓練和資料管道以及推理。

技術概述

本節重點介紹NetApp的 OpenSource MLOps 技術概述。

人工智慧

人工智慧是一門電腦科學學科，其中電腦經過訓練可以模仿人類思維的認知功能。人工智慧開發人員訓練電腦以類似於人類甚至優於人類的方式學習和解決問題。深度學習和機器學習是人工智慧的子領域。越來越多的組織採用 AI、ML 和 DL 來支援其關鍵業務需求。以下是一些範例：

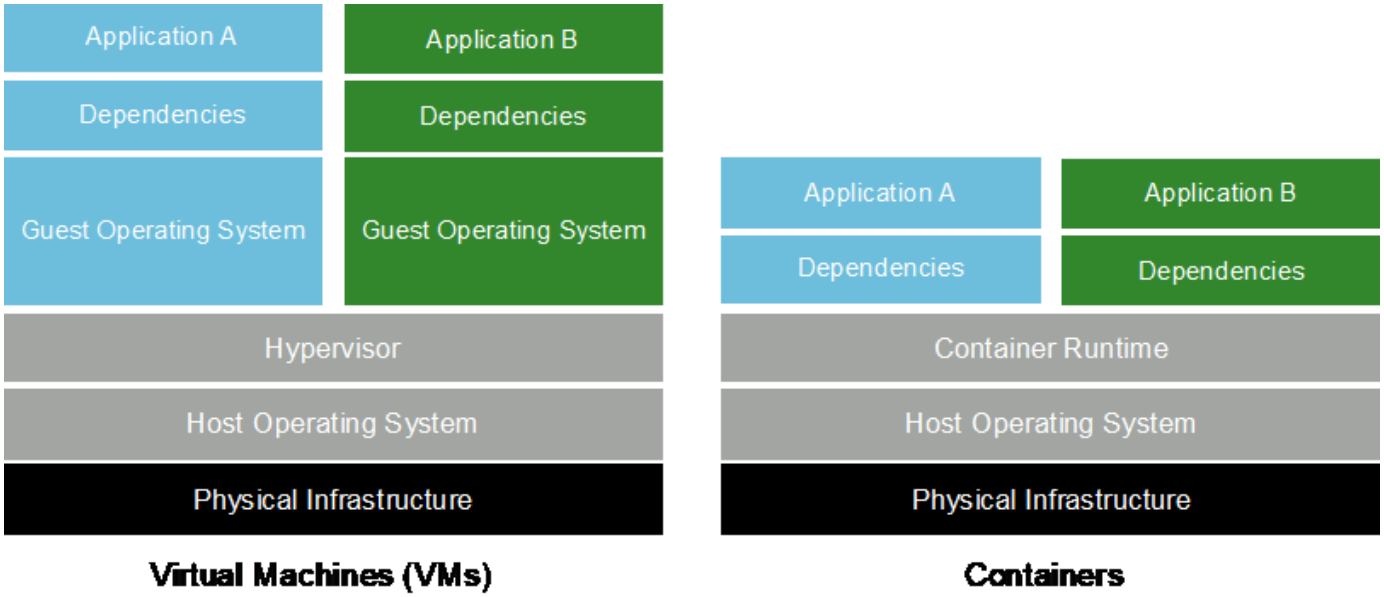
- 分析大量數據以發掘以前未知的商業見解
- 使用自然語言處理直接與客戶互動
- 自動化各種業務流程與功能

現代人工智慧訓練和推理工作負載需要大規模並行運算能力。因此，GPU 越來越多地被用於執行 AI 操作，因為 GPU 的平行處理能力遠遠優於通用 CPU。

容器

容器是在共享主機作業系統核心上運行的隔離的使用者空間實例。容器的採用正在迅速增加。容器提供許多與虛擬機器 (VM) 相同的應用程式沙盒優勢。然而，由於虛擬機器所依賴的虛擬機器管理程式和客戶作業系統層已被消除，因此容器變得更加輕量級。下圖描述了虛擬機器與容器的可視化。

容器還允許直接將應用程式依賴項、運行時間等與應用程式有效率地打包在一起。最常用的容器打包格式是Docker容器。以 Docker 容器格式容器化的應用程式可以在任何可以執行 Docker 容器的機器上執行。即使應用程式的依賴項不存在於機器上，情況也是如此，因為所有依賴項都打包在容器本身中。欲了解更多信息，請訪問 "Docker 網站"。



Kubernetes

Kubernetes 是一個開源的、分散式的容器編排平台，最初由 Google 設計，現在由雲端原生運算基金會 (CNCF) 維護。 Kubernetes 支援容器化應用程式的部署、管理和擴充功能的自動化。近年來，Kubernetes 已成為主流的容器編排平台。欲了解更多信息，請訪問 "Kubernetes 網站"。

NetApp Trident

"Trident"支援在所有流行的NetApp儲存平台上（公有雲或內部）使用和管理儲存資源，包括ONTAP （AFF、FAS、Select、Cloud、 Amazon FSx ONTAP）、 Azure NetApp Files服務和Google Cloud NetApp Volumes。Trident是一個符合容器儲存介面 (CSI) 的動態儲存編排器，可與 Kubernetes 原生整合。

NetApp DataOps 工具包

這"NetApp DataOps 工具包"是一個基於 Python 的工具，可簡化由高效能、橫向擴展NetApp儲存支援的開發/培訓工作區和推理伺服器的管理。主要功能包括：

- 快速配置由高效能、橫向擴展NetApp儲存支援的新的高容量工作區。
- 近乎即時地克隆高容量工作區，以實現實驗或快速迭代。
- 幾乎即時地保存高容量工作區的快照，以用於備份和/或可追溯性/基準測試。
- 近乎即時地提供、複製和快照大容量、高效能資料磁碟區。

Apache Airflow

Apache Airflow 是一個開源工作流程管理平台，支援以程式設計方式編寫、排程和監控複雜的企業工作流程。它通常用於自動化 ETL 和資料管道工作流程，但並不局限於這些類型的工作流程。 Airflow 計畫由 Airbnb 發起，但後來在業界變得非常流行，現在由 Apache 軟體基金會贊助。 Airflow 是用 Python 編寫的，Airflow 工作流程是透過 Python 腳本建立的，而 Airflow 是在「配置即程式碼」的原則下設計的。許多企業 Airflow 用戶現在在 Kubernetes 上運行 Airflow。

有向無環圖（**DAG**）

在 Airflow 中，工作流程被稱為有向無環圖 (DAG)。 DAG 由依序、並行或兩者結合執行的任務組成，取決於 DAG 定義。 Airflow 排程器在一組工作器上執行各個任務，遵守 DAG 定義中指定的任務層級依賴關係。 DAG 是透過 Python 腳本定義和建立的。

Jupyter 筆記本

Jupyter Notebooks 是類似 wiki 的文檔，包含即時程式碼和描述性文字。 Jupyter Notebooks 在 AI 和 ML 社群中被廣泛用作記錄、儲存和共享 AI 和 ML 專案的一種方式。有關 Jupyter Notebooks 的更多信息，請訪問 "Jupyter 網站"。

Jupyter Notebook 伺服器

Jupyter Notebook 伺服器是一個開源 Web 應用程序，允許使用者建立 Jupyter Notebook。

JupyterHub

JupyterHub 是一個多用戶應用程序，允許個人用戶配置和存取自己的 Jupyter Notebook 伺服器。有關 JupyterHub 的更多信息，請訪問 "JupyterHub 網站"。

機器學習流

MLflow 是一個流行的開源 AI 生命週期管理平台。 MLflow 的主要功能包括 AI/ML 實驗追蹤和 AI/ML 模型庫。有關 MLflow 的更多信息，請訪問 "MLflow 網站"。

Kubeflow

Kubeflow 是 Kubernetes 的開源 AI 和 ML 工具包，最初由 Google 開發。 Kubeflow 專案使得在 Kubernetes 上部署 AI 和 ML 工作流程變得簡單、可移植且可擴充。 Kubeflow 抽象化了 Kubernetes 的複雜性，使資料科學家能夠專注於他們最了解的領域——資料科學。請參考下圖以了解視覺化效果。對於喜歡一體化 MLOps 平台的組織來說，Kubeflow 是一個不錯的開源選擇。欲了解更多信息，請訪問 "Kubeflow 網站"。

Kubeflow 管道

Kubeflow Pipelines 是 Kubeflow 的關鍵元件。 Kubeflow Pipelines 是一個用於定義和部署可移植、可擴展的 AI 和 ML 工作流程的平台和標準。有關詳細信息，請參閱 "Kubeflow 官方文檔"。

Kubeflow 筆記本

Kubeflow 簡化了 Kubernetes 上 Jupyter Notebook 伺服器的設定和部署。有關 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，請參閱 "Kubeflow 官方文檔"。

卡提布

Katib 是一個用於自動化機器學習 (AutoML) 的 Kubernetes 原生專案。 Katib 支援超參數調整、早期停止和神經架構搜尋 (NAS)。 Katib 是一個與機器學習 (ML) 框架無關的專案。它可以調整使用者選擇的任何語言編寫的應用程式的超參數，並且原生支援許多 ML 框架，例如 TensorFlow、MXNet、PyTorch、XGBoost 等。 Katib 支援許多不同的 AutoML 演算法，例如貝葉斯優化、Parzen 估計器樹、隨機搜尋、協方差矩陣自適應進化策略、超頻、高效能神經架構搜尋、可微分架構搜尋等等。有關 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，請參閱 "Kubeflow 官方文檔"。

NetApp ONTAP

ONTAP 9 是NetApp最新一代儲存管理軟體，它支援企業實現基礎架構現代化並過渡到雲端就緒資料中心。ONTAP利用業界領先的數據管理功能，只需一套工具即可管理和保護數據，無論數據位於何處。您也可以將資料自由移動到任何需要的地方：邊緣、核心或雲端。 ONTAP 9 包含眾多功能，可簡化資料管理、加速和保護關鍵數據，並支援跨混合雲架構的下一代基礎架構功能。

簡化資料管理

資料管理對於企業 IT 營運和資料科學家至關重要，以便將適當的資源用於 AI 應用程式和訓練 AI/ML 資料集。以下有關NetApp技術的附加資訊超出了本次驗證的範圍，但可能與您的部署相關。

ONTAP資料管理軟體包括以下功能，可簡化操作並降低總營運成本：

- 內聯資料壓縮和擴展重複資料刪除。資料壓縮減少了儲存區塊內部浪費的空間，重複資料刪除顯著增加了有效容量。這適用於本地儲存的資料和分層到雲端的資料。
- 最小、最大和自適應服務品質 (AQoS)。細粒度的服務品質 (QoS) 控制有助於維持高度共享環境中關鍵應用程式的效能水準。
- NetApp FabricPool。提供冷資料到公有和私有雲儲存選項的自動分層，包括 Amazon Web Services (AWS)、Azure 和NetApp StorageGRID儲存解決方案。有關FabricPool的更多信息，請參閱 "TR-4598：FabricPool最佳實踐"。

加速並保護數據

ONTAP提供卓越等級的效能和資料保護，並透過以下方式擴展這些功能：

- 性能和更低的延遲。 ONTAP以盡可能低的延遲提供盡可能高的吞吐量。
- 資料保護。 ONTAP提供內建資料保護功能，並在所有平台上提供通用管理。
- NetApp磁碟區加密 (NVE)。 ONTAP提供原生磁碟區級加密，同時支援板載和外部金鑰管理。
- 多租戶和多因素身份驗證。 ONTAP支援以最高等級的安全性共用基礎架構資源。

面向未來的基礎設施

ONTAP具有以下功能，可協助滿足嚴苛且不斷變化的業務需求：

- 無縫擴展和無中斷運行。 ONTAP支援無中斷地向現有控制器和橫向擴展叢集添加容量。客戶可以升級到最新技術，而無需昂貴的資料遷移或中斷。
- 雲端連線。 ONTAP是與雲端連線最緊密的儲存管理軟體，在所有公有雲中提供軟體定義儲存和雲端原生執行個體的選項。
- 與新興應用程式的整合。 ONTAP使用支援現有企業應用的相同基礎架構，為下一代平台和應用（如自動駕駛汽車、智慧城市和工業 4.0）提供企業級資料服務。

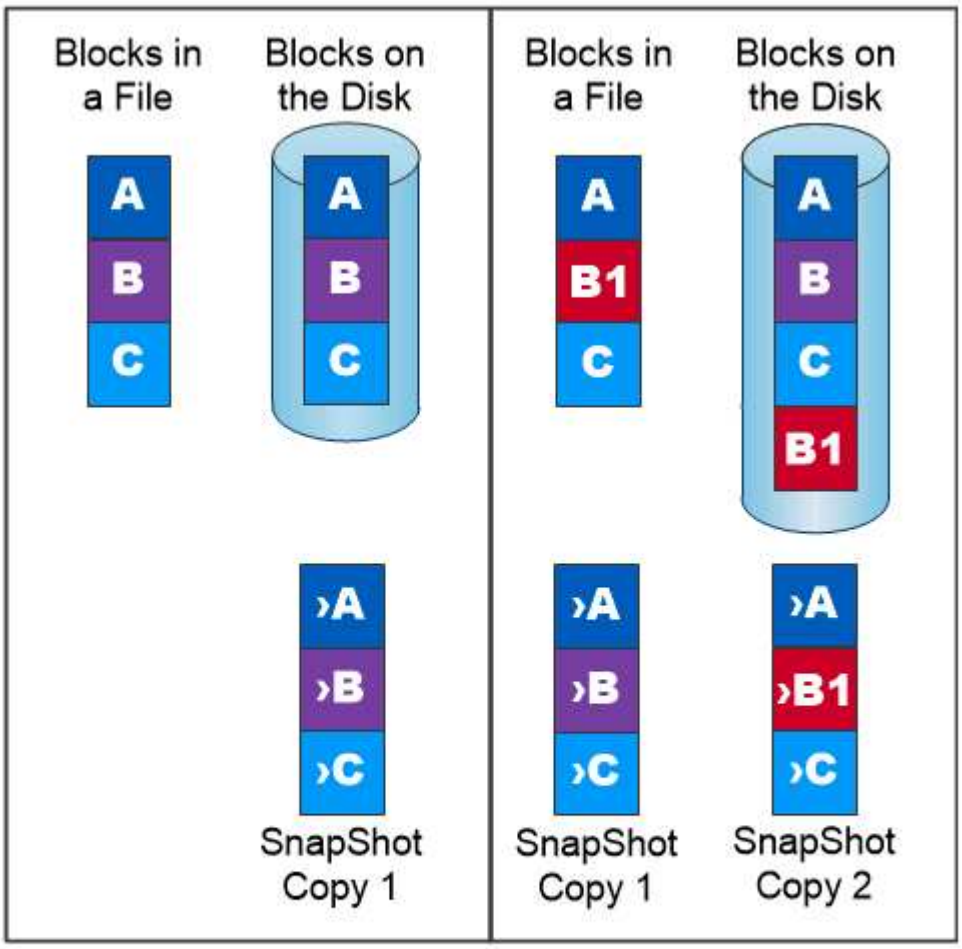
NetApp快照副本

NetApp Snapshot 副本是磁碟區的唯讀、時間點映像。該影像佔用的儲存空間極小，且產生的效能開銷可以忽略不計，因為它僅記錄自上次 Snapshot 副本建立以來對檔案的更改，如下圖所示。

Snapshot 副本的效率歸功於核心ONTAP儲存虛擬化技術，即任意位置寫入檔案佈局 (WAFL)。與資料庫一樣，WAFL使用元資料指向磁碟上的實際資料區塊。但是，與資料庫不同， WAFL不會覆蓋現有區塊。它將更新的資

料寫入新區塊並更改元資料。這是因為ONTAP在創建 Snapshot 副本時引用元數據，而不是複製數據塊，所以 Snapshot 副本非常有效率。這樣做可以消除其他系統在定位要複製的區塊時產生的尋道時間，以及複製本身的成本。

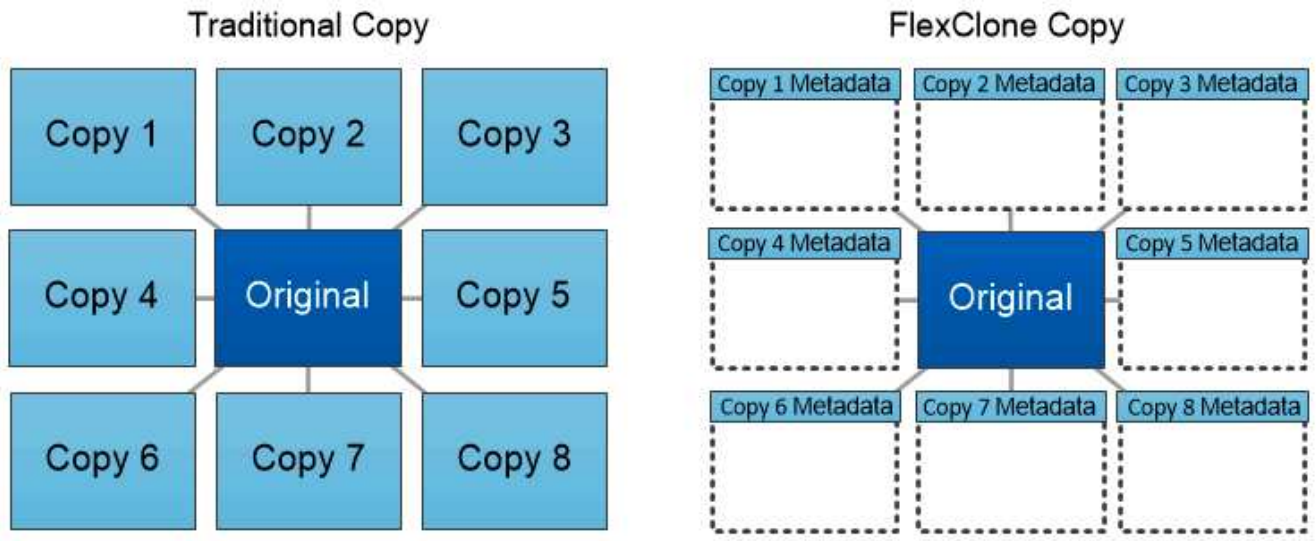
您可以使用 Snapshot 副本來還原單一檔案或 LUN，或還原磁碟區的全部內容。 ONTAP將 Snapshot 副本中的指標資訊與磁碟上的資料進行比較，以重建遺失或損壞的對象，而無需停機或造成顯著的效能成本。



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

NetApp FlexClone技術

NetApp FlexClone技術參考 Snapshot 元資料來建立磁碟區的可寫入時間點副本。副本與其父級共用資料塊，除了元資料所需的儲存空間外，不消耗任何儲存空間，直到將變更寫入副本為止，如下圖所示。傳統的複製可能需要幾分鐘甚至幾小時才能創建，而FlexClone軟體可以讓您幾乎立即複製最大的資料集。這使得它非常適合需要相同資料集的多個副本（例如，開發工作區）或資料集的臨時副本（針對生產資料集測試應用程式）的情況。



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

NetApp SnapMirror資料複製技術

NetApp SnapMirror軟體是一種跨資料結構的經濟高效、易於使用的統一複製解決方案。它透過 LAN 或 WAN 高速複製資料。它為所有類型的應用程式（包括虛擬和傳統環境中的關鍵業務應用程式）提供高資料可用性和快速資料複製。當您將資料複製到一個或多個NetApp儲存系統並不斷更新輔助資料時，您的資料將保持最新狀態並可隨時使用。不需要外部複製伺服器。下圖是利用SnapMirror技術的架構範例。

SnapMirror軟體透過網路僅發送更改的區塊來利用NetApp ONTAP儲存效率。 SnapMirror軟體還使用內建網路壓縮來加速資料傳輸並將網路頻寬利用率降低高達 70%。借助SnapMirror技術，您可以利用一個精簡複製資料流來建立一個儲存庫，同時維護活動鏡像和先前的時間點副本，從而將網路流量減少高達 50%。

NetApp BlueXP複製與同步

"BlueXP複製和同步"是NetApp 的一項快速、安全的資料同步服務。無論您需要在本機 NFS 或 SMB 檔案共用、NetApp StorageGRID、 NetApp ONTAP S3、 Google Cloud NetApp Volumes、 Azure NetApp Files、AWS S3、AWS EFS、Azure Blob、Google Cloud Storage 或 IBM CloudObject Storage 之間傳輸檔案， BlueXP Copy and Sync 都能快速安全地移動到您的位置。

資料傳輸完成後，可在來源端和目標端完全使用。 BlueXP Copy and Sync 可以在觸發更新時按需同步數據，或根據預先定義的時間表連續同步數據。無論如何， BlueXP Copy and Sync 僅移動增量，因此在資料複製上花費的時間和金錢被最小化。

BlueXP Copy and Sync 是一種軟體即服務 (SaaS) 工具，其設定和使用極為簡單。 BlueXP Copy 和 Sync 觸發的資料傳輸由資料代理執行。 BlueXP Copy 和 Sync 資料代理程式可以部署在 AWS、Azure、Google Cloud Platform 或本地端。

NetApp XCP

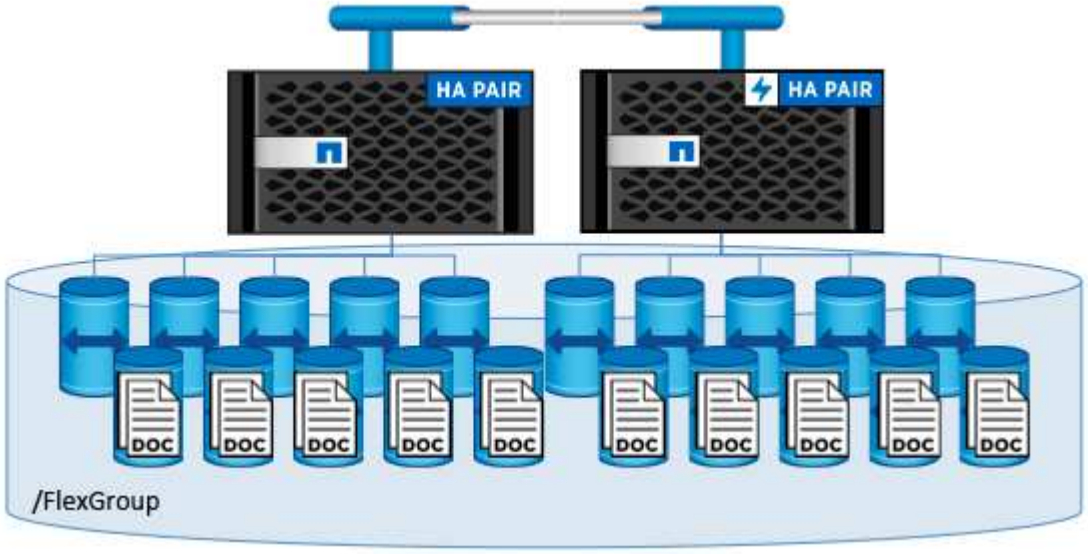
"NetApp XCP"是一款基於客戶端的軟體，用於任意到NetApp和NetApp到NetApp 的資料遷移和檔案系統洞察。XCP 旨在透過利用所有可用的系統資源來處理大容量資料集和高效能遷移，從而實現擴展並實現最大效能。XCP 可協助您全面了解檔案系統，並提供產生報告的選項。

NetApp ONTAP FlexGroup卷

訓練資料集可能包含數十億個檔案。文件可以包括文字、音訊、視訊和其他形式的非結構化數據，這些數據必須儲存和處理才能並行讀取。儲存系統必須儲存大量小文件，並且必須並行讀取這些文件以實現順序和隨機 I/O。

FlexGroup磁碟區是一個由多個組成成員磁碟區組成的單一命名空間，如下圖所示。從儲存管理員的角度來看，FlexGroup磁碟區的管理和行為類似於NetApp FlexVol volume。 FlexGroup卷中的檔案被指派給各個成員卷，並且不會跨卷或節點進行條帶化。它們支援以下功能：

- FlexGroup磁碟區為高元資料工作負載提供了數 PB 的容量和可預測的低延遲。
- 它們支援同一命名空間中最多 4000 億個檔案。
- 它們支援跨 CPU、節點、聚合體和組成FlexVol磁碟區的 NAS 工作負載的平行操作。



架構

該解決方案不依賴特定的硬體。此解決方案與NetApp Trident支援的任何NetApp實體儲存設備、軟體定義實例或雲端服務相容。範例包括NetApp AFF儲存系統、 Amazon FSx ONTAP、 Azure NetApp Files、 Google Cloud NetApp Volumes或NetApp Cloud Volumes ONTAP個體。此外，只要所使用的 Kubernetes 版本受到NetApp Trident和正在實施的其他解決方案元件的支持，該解決方案就可以在任何 Kubernetes 叢集上實作。有關Trident支援的 Kubernetes 版本列表，請參閱 "Trident文檔"。有關用於驗證此解決方案的各個組件的環境的詳細信息，請參閱下表。

Apache Airflow 驗證環境

軟體元件	版本
Apache Airflow	2.0.1，透過以下方式部署"Apache Airflow Helm 圖表 "8.0.8
Kubernetes	1.18

軟體元件	版本
NetApp Trident	21.01

JupyterHub 驗證環境

軟體元件	版本
JupyterHub	4.1.5，透過部署"JupyterHub Helm 圖表"3.3.7
Kubernetes	1.29
NetApp Trident	24.02

MLflow 驗證環境

軟體元件	版本
機器學習流	2.14.1，透過部署"MLflow Helm 圖表"1.4.12
Kubernetes	1.29
NetApp Trident	24.02

Kubeflow 驗證環境

軟體元件	版本
Kubeflow	1.7，透過部署"部署KF"0.1.1
Kubernetes	1.26
NetApp Trident	23.07

支援

NetApp不為 Apache Airflow、JupyterHub、MLflow、Kubeflow 或 Kubernetes 提供企業支援。如果您對完全支援的 MLOps 平台感興趣，"聯絡NetApp"了解NetApp與合作夥伴共同提供的全面支援的 MLOps 解決方案。

NetApp Trident配置

NetApp AIPod部署的Trident後端範例

在使用Trident在 Kubernetes 叢集中動態設定儲存資源之前，您必須建立一個或多個Trident後端。以下範例代表如果您要在下列位置部署此解決方案的元件，您可能需要建立的不同類型的後端："NetApp AIPod" 。有關後端的更多信息，以及其他平台/環境的後端示例，請參閱"Trident文檔"。

1. NetApp建議為您的AIPod建立支援FlexGroup的Trident Backend。

   以下的範例指令展示如何為AIPod儲存虛擬機器 (SVM) 建立支援FlexGroup的Trident Backend。此後端使用 `ontap-nas-flexgroup`儲存驅動程式。 ONTAP支援兩種主要資料磁碟區類型： FlexVol和FlexGroup。

FlexVol卷的大小受到限制（截至撰寫本文時，最大大小取決於具體的部署）。另一方面， FlexGroup磁碟區可以線性擴展到最多 20PB 和 4000 億個文件，從而提供單一命名空間，大大簡化資料管理。因此， FlexGroup磁碟區最適合依賴大量資料的 AI 和 ML 工作負載。

如果您處理的資料量較小，並且想要使用FlexVol捲而不是FlexGroup卷，則可以建立使用 `ontap-nas`儲存驅動程序，而不是 `ontap-nas-flexgroup`儲存驅動程式。

```
$ 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也建議建立支援FlexVol的Trident Backend。您可能希望使用FlexVol磁碟區來託管持久性應用程式、儲存結果、輸出、偵錯資訊等。如果要使用FlexVol磁碟區，則必須建立一個或多個啟用FlexVol的Trident後端。下面的範例指令顯示如何建立啟用單一FlexVol的Trident後端。

```
$ 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部署的 Kubernetes 儲存類別範例

在使用Trident在 Kubernetes 叢集中動態設定儲存資源之前，您必須建立一個或多個 Kubernetes StorageClasses。以下範例代表如果您在下列位置部署此解決方案的元件，您可能需要建立的不同類型的 StorageClasses："NetApp AIPod" 。有關 StorageClasses 的更多信息，以及其他平台/環境的 StorageClasses 範例，請參閱"Trident文檔"。

1. NetApp建議為您在本節中建立的支援FlexGroup的Trident Backend 建立 StorageClass"NetApp AIPod部署的Trident後端範例"中，步驟 1。下面的範例指令顯示如何建立多個 StorageClasses，這些 StorageClasses 與本節中建立的範例 Backend 相對應"NetApp AIPod部署的Trident後端範例"，步驟 1 - 利用"基於 RDMA 的 NFS"還有一個則不然。

   為了確保持久卷在刪除相應的 PersistentVolumeClaim (PVC) 時不會被刪除，以下範例使用 `reclaimPolicy`的價值 `Retain`。有關 `reclaimPolicy`字段，請參閱官方 "Kubernetes 文檔"。

   注意：下列範例 StorageClasses 使用的最大傳輸大小為 262144。若要使用此最大傳輸大小，您必須在ONTAP系統上相應地配置最大傳輸大小。請參閱"ONTAP 文件"了解詳情。

   注意：要使用 NFS over RDMA，您必須在ONTAP系統上設定 NFS over RDMA。請參閱"ONTAP 文件"了解詳情。

   注意：在以下範例中，StorageClass 定義檔中的 storagePool 欄位指定了具體的 Backend。

```
$ 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也建議建立一個與您在本節中建立的支援FlexVol的Trident Backend 相對應的 StorageClass"用於AIPod部署的Trident後端範例"中，步驟 2。下面的範例指令展示如何為FlexVol磁碟區建立單一StorageClass。

   注意：在下面的範例中，StorageClass 定義檔中的 storagePool 欄位未指定特定的 Backend。當你使用 Kubernetes 來管理使用此 StorageClass 的磁碟區時， Trident會嘗試使用任何可用的後端，該後端使用 `ontap-nas`司機。

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

Apache Airflow

Apache Airflow 部署

本節介紹在 Kubernetes 叢集中部署 Airflow 必須完成的任務。

> 可以在 Kubernetes 以外的平台上部署 Airflow。在 Kubernetes 以外的平台上部署 Airflow 超出了本解決方案的範圍。

先決條件

在執行本節概述的部署練習之前，我們假設您已經執行了以下任務：

1. 您已經有一個可以運行的 Kubernetes 叢集。
2. 您已經在 Kubernetes 叢集中安裝並設定了NetApp Trident 。有關Trident的更多詳細信息，請參閱"Trident文檔"。

安裝 **Helm**

Airflow 使用 Helm（Kubernetes 的流行套件管理器）進行部署。在部署 Airflow 之前，必須在部署跳轉主機上安裝 Helm。若要在部署跳轉主機上安裝 Helm，請依照 "安裝說明"在 Helm 官方文件中。

設定預設 **Kubernetes StorageClass**

在部署 Airflow 之前，您必須在 Kubernetes 叢集中指定一個預設 StorageClass。 Airflow 部署程序嘗試使用預設 StorageClass 來設定新的持久性磁碟區。如果沒有指定 StorageClass 作為預設 StorageClass，則部署失敗。若要在叢集中指定預設 StorageClass，請依照"Kubeflow部署"部分。如果您已經在叢集中指定了預設 StorageClass，則可以跳過此步驟。

使用 **Helm** 部署 **Airflow**

若要使用 Helm 在 Kubernetes 叢集中部署 Airflow，請從部署跳轉主機執行下列任務：

1. 按照以下說明使用 Helm 部署 Airflow "部署說明"查看 Artifact Hub 上的官方 Airflow 圖表。下面的範例指令展示了使用 Helm 部署 Airflow。修改、新增和/或刪除 `custom- values.yaml`根據您的環境和所需配置，根據需要建立文件。

```
$ 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
```

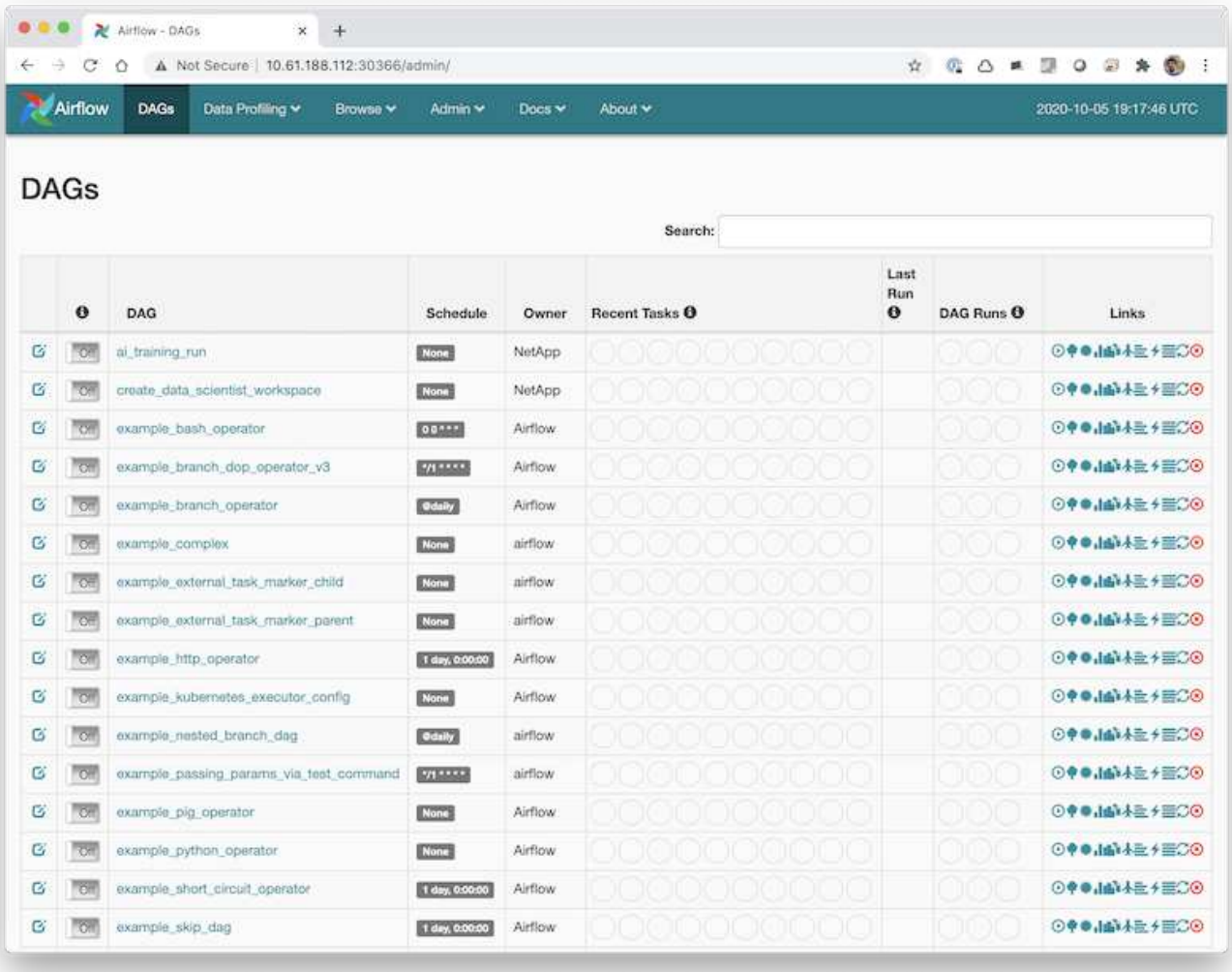
2. 確認所有 Airflow pod 均已啟動並正在運作。所有 pod 啟動可能需要幾分鐘。

```
$ 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. 請依照步驟 1 中使用 Helm 部署 Airflow 時列印到控制台的說明取得 Airflow Web 服務 URL。

```
$ 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. 確認您可以存取 Airflow Web 服務。



將NetApp DataOps 工具包與 Airflow 結合使用

這 "適用於 Kubernetes 的NetApp DataOps 工具包"可以與 Airflow 結合使用。將NetApp DataOps Toolkit 與 Airflow 結合使用，您可以將NetApp資料管理作業（例如建立快照和複製）合併到由 Airflow 協調的自動化工作流程中。

請參閱 "氣流範例"有關將該工具包與 Airflow 結合使用的詳細信息，請參閱NetApp DataOps Toolkit GitHub 儲存庫中的部分。

JupyterHub

JupyterHub 部署

本節介紹在 Kubernetes 叢集中部署 JupyterHub 必須完成的任務。

> 可以在 Kubernetes 以外的平台上部署 JupyterHub。在 Kubernetes 以外的平台上部署 JupyterHub 超出了本解決方案的範圍。

先決條件

在執行本節概述的部署練習之前，我們假設您已經執行了以下任務：

1. 您已經有一個可以運行的 Kubernetes 叢集。
2. 您已經在 Kubernetes 叢集中安裝並設定了NetApp Trident 。有關Trident的更多詳細信息，請參閱"Trident文檔"。

安裝 **Helm**

JupyterHub 使用 Helm（Kubernetes 的熱門套件管理器）進行部署。在部署 JupyterHub 之前，您必須在 Kubernetes 控制節點上安裝 Helm。要安裝 Helm，請按照 "安裝說明"在 Helm 官方文件中。

設定預設 **Kubernetes StorageClass**

在部署 JupyterHub 之前，您必須在 Kubernetes 叢集中指定一個預設 StorageClass。若要在叢集中指定預設 StorageClass，請依照"Kubeflow部署"部分。如果您已經在叢集中指定了預設 StorageClass，則可以跳過此步驟。

部署 **JupyterHub**

完成上述步驟後，現在可以部署 JupyterHub 了。 JupyterHub 部署需要以下步驟：

設定 **JupyterHub** 部署

在部署之前，最好先針對各自的環境最佳化 JupyterHub 部署。您可以建立一個 **config.yaml** 檔案並在使用 Helm 圖表部署期間使用它。

可以在以下位置找到範例 **config.yaml** 文件 https://github.com/jupyterhub/zero-to-jupyterhub-k8s/blob/HEAD/jupyterhub/values.yaml

> 在此 config.yaml 檔案中，您可以為NetApp Trident StorageClass 設定 **(singleuser.storage.dynamic.storageClass)** 參數。這是用於為各個使用者工作區配置磁碟區的儲存類別。

新增共享磁碟區

如果您想要為所有 JupyterHub 使用者使用共享卷，您可以相應地調整您的 **config.yaml**。例如，如果您有一個名為 jupyterhub-shared-volume 的共用 PersistentVolumeClaim，則可以將其作為 /home/shared 掛載在所有使用者 pod 中，如下所示：

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

這是可選步驟，您可以根據需要調整這些參數。

使用 **Helm Chart** 部署 **JupyterHub**

讓 Helm 了解 JupyterHub Helm 圖表儲存庫。

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

這應該會顯示如下輸出：

```
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!□
```

現在透過從包含您的 config.yaml 的目錄執行以下命令來安裝由您的 config.yaml 設定的圖表：

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

在此範例中：

<helm-release-name> 設定為 my-jupyterhub，這將是您的 JupyterHub 版本的名稱。 <k8s-namespace> 設定為 my-namespace，這是您要安裝 JupyterHub 的命名空間。如果命名空間不存在，則使用 --create-namespace 標誌建立命名空間。 --values 標誌指定包含所需設定選項的 config.yaml 檔案。

檢查部署

在步驟 2 運行時，您可以透過以下命令看到正在建立的 pod：

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

等待 hub 和 proxy pod 進入 Running 狀態。

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

造訪 **JupyterHub**

尋找我們可以用來存取 JupyterHub 的 IP。執行以下命令，直到代理公共服務的 EXTERNAL-IP 可用，如範例輸出所示。

我們在 config.yaml 檔案中使用了 NodePort 服務，您可以根據您的設定（例如 LoadBalancer）調整您的環境。

```
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
```

若要使用 JupyterHub，請在瀏覽器中輸入代理公共服務的外部 IP。

將**NetApp DataOps** 工具包與 **JupyterHub** 結合使用

這 "適用於 Kubernetes 的NetApp DataOps 工具包"可與 JupyterHub 結合使用。透過將NetApp DataOps Toolkit 與 JupyterHub 結合使用，最終使用者可以直接在 Jupyter Notebook 中建立用於工作區備份和/或資料集到模型可追溯性的磁碟區快照。

初始設定

在將 DataOps Toolkit 與 JupyterHub 一起使用之前，您必須向 JupyterHub 指派給各個使用者 Jupyter Notebook Server pod 的 Kubernetes 服務帳戶授予適當的權限。 JupyterHub 使用由 `singleuser.serviceAccountName`JupyterHub Helm 圖表設定檔中的變數。

為 **DataOps Toolkit** 建立叢集角色

首先，建立一個名為「netapp-dataops」的叢集角色，該角色具有建立磁碟區快照所需的 Kubernetes API 權

限。

```
$ 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
```

將叢集角色指派給筆記本伺服器服務帳戶

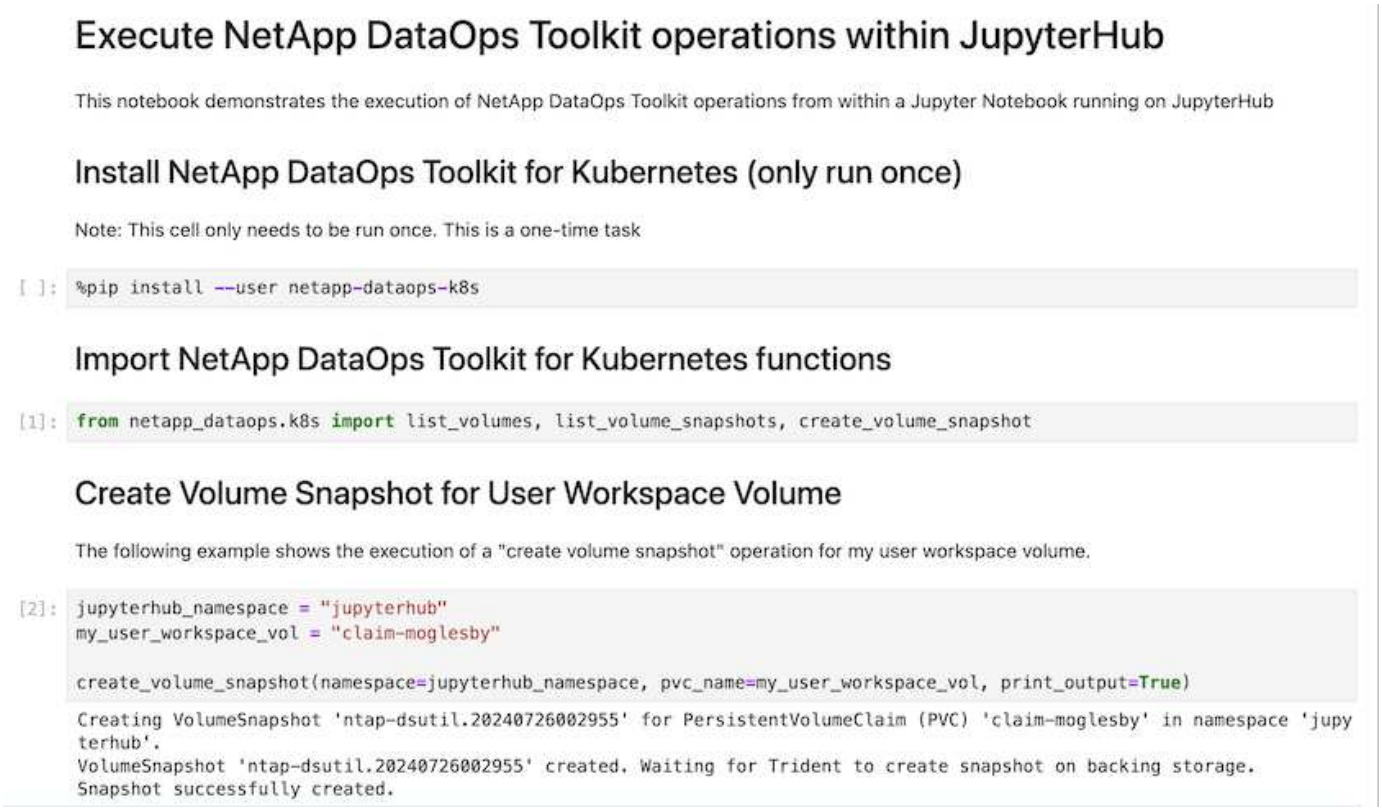
建立一個角色綁定，將「netapp-dataops-snapshots」叢集角色指派給適當命名空間中的適當服務帳戶。例如，如果您在「jupyterhub」命名空間中安裝了 JupyterHub，並且透過以下方式指定了「預設」服務帳戶 `singleuser.serviceAccountName`變量，您需要將“netapp-dataops-snapshots”集群角色指派給“jupyterhub”命名空間中的“預設”服務帳戶，如下例所示。

```
$ 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
```

在 **Jupyter Notebook** 中建立卷宗快照

現在，JupyterHub 使用者可以使用NetApp DataOps Toolkit 直接從 Jupyter Notebook 建立磁碟區快照，如下例所示。



使用**NetApp SnapMirror**將資料匯入 **JupyterHub**

NetApp SnapMirror是一種複製技術，可讓您在NetApp儲存系統之間複製資料。SnapMirror可用於將遠端環境中的資料提取到 JupyterHub。

範例工作流程和演示

參考"此 Tech ONTAP部落格文章"有關使用NetApp SnapMirror將資料匯入 JupyterHub 的詳細範例工作流程和示範。

機器學習流

MLflow部署

本節介紹在 Kubernetes 叢集中部署 MLflow 必須完成的任務。

> 可以在 Kubernetes 以外的平台上部署 MLflow。在 Kubernetes 以外的平台上部署 MLflow 超出了本解決方案的範圍。

先決條件

在執行本節概述的部署練習之前，我們假設您已經執行了以下任務：

1. 您已經有一個可以運行的 Kubernetes 叢集。
2. 您已經在 Kubernetes 叢集中安裝並設定了NetApp Trident 。有關Trident的更多詳細信息，請參閱"Trident文檔"。

安裝 **Helm**

MLflow 使用 Helm（Kubernetes 的流行套件管理器）進行部署。在部署 MLflow 之前，必須在 Kubernetes 控制節點上安裝 Helm。要安裝 Helm，請按照 "安裝說明"在 Helm 官方文件中。

設定預設 **Kubernetes StorageClass**

在部署 MLflow 之前，您必須在 Kubernetes 叢集中指定一個預設 StorageClass。若要在叢集中指定預設 StorageClass，請依照"Kubeflow部署"部分。如果您已經在叢集中指定了預設 StorageClass，則可以跳過此步驟。

部署 **MLflow**

滿足先決條件後，您就可以使用 Helm Chart 開始 MLflow 部署。

設定 **MLflow Helm Chart** 部署。

在使用 Helm 圖表部署 MLflow 之前，我們可以使用 **config.yaml** 檔案將部署配置為使用NetApp Trident儲存類別並更改其他參數以滿足我們的需求。您可以在以下位置找到 **config.yaml** 檔案的範例： https://github.com/bitnami/charts/blob/main/bitnami/mlflow/values.yaml

> 您可以在 config.yaml 檔案中的 **global.defaultStorageClass** 參數下設定Trident storageClass（例如 storageClass：「ontap-flexvol」）。

安裝 **Helm Chart**

可以使用以下命令將 Helm 圖表與 MLflow 的自訂 **config.yaml** 檔案一起安裝：

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

> 該命令透過提供的*config.yaml*檔案在自訂配置中的 Kubernetes 叢集上部署 MLflow。 MLflow 部署在給定的命名空間中，並透過 kubernetes 為該版本提供一個隨機發布名稱。

檢查部署

Helm 圖表部署完成後，您可以使用以下命令檢查服務是否可存取：

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

> 將 **jupyterhub** 替換為您在部署期間使用的命名空間。

您應該會看到以下服務：

```
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
```

我們編輯了 config.yaml 檔案以使用 NodePort 服務存取連接埠 30002 上的 MLflow。

存取 **MLflow**

一旦與 MLflow 相關的所有服務都啟動並運行，您就可以使用給定的 NodePort 或 LoadBalancer IP 位址存取它（例如 `http://10.61.181.109:30002`）

使用**NetApp**和 **MLflow** 實現資料集到模型的可追溯性

這 "適用於 Kubernetes 的NetApp DataOps 工具包"可以與 MLflow 的實驗追蹤功能結合使用，以實現資料集到模型或工作區到模型的可追溯性。

要實現資料集到模型或工作區到模型的可追溯性，只需在訓練運行過程中使用 DataOps Toolkit 建立資料集或工作區磁碟區的快照，如下列範例程式碼片段所示。此程式碼將資料卷名稱和快照名稱儲存為與您記錄到 MLflow 實驗追蹤伺服器的特定訓練運行相關的標籤。

```
...
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)

    ...
```

Kubeflow

Kubeflow部署

本節介紹在 Kubernetes 叢集中部署 Kubeflow 必須完成的任務。

先決條件

在執行本節概述的部署練習之前，我們假設您已經執行了以下任務：

1. 您已經有一個可運行的 Kubernetes 集群，並且您正在運行您打算部署的 Kubeflow 版本支援的 Kubernetes 版本。有關支援的 Kubernetes 版本列表，請參閱 Kubeflow 版本的依賴項"Kubeflow 官方文檔"。
2. 您已經在 Kubernetes 叢集中安裝並設定了NetApp Trident 。有關Trident的更多詳細信息，請參閱"Trident文檔"。

設定預設 **Kubernetes StorageClass**

在部署 Kubeflow 之前，我們建議在 Kubernetes 叢集中指定一個預設 StorageClass。 Kubeflow 部署程序可能會嘗試使用預設 StorageClass 來設定新的持久性磁碟區。如果沒有指定 StorageClass 作為預設 StorageClass

，則部署可能會失敗。若要在叢集中指定預設 StorageClass，請從部署跳轉主機執行下列任務。如果您已經在叢集中指定了預設 StorageClass，則可以跳過此步驟。

1. 將現有 StorageClass 之一指定為預設 StorageClass。以下範例命令顯示了名為 `ontap-ai-flexvols-retain`作為預設的 StorageClass。

> 這 `ontap-nas-flexgroup`Trident Backend 類型的最小 PVC 尺寸相當大。預設情況下，Kubeflow 嘗試配置大小僅為幾 GB 的 PVC。因此，您不應該指定使用 `ontap-nas-flexgroup`後端類型作為 Kubeflow 部署的預設 StorageClass。

```
$ 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
```

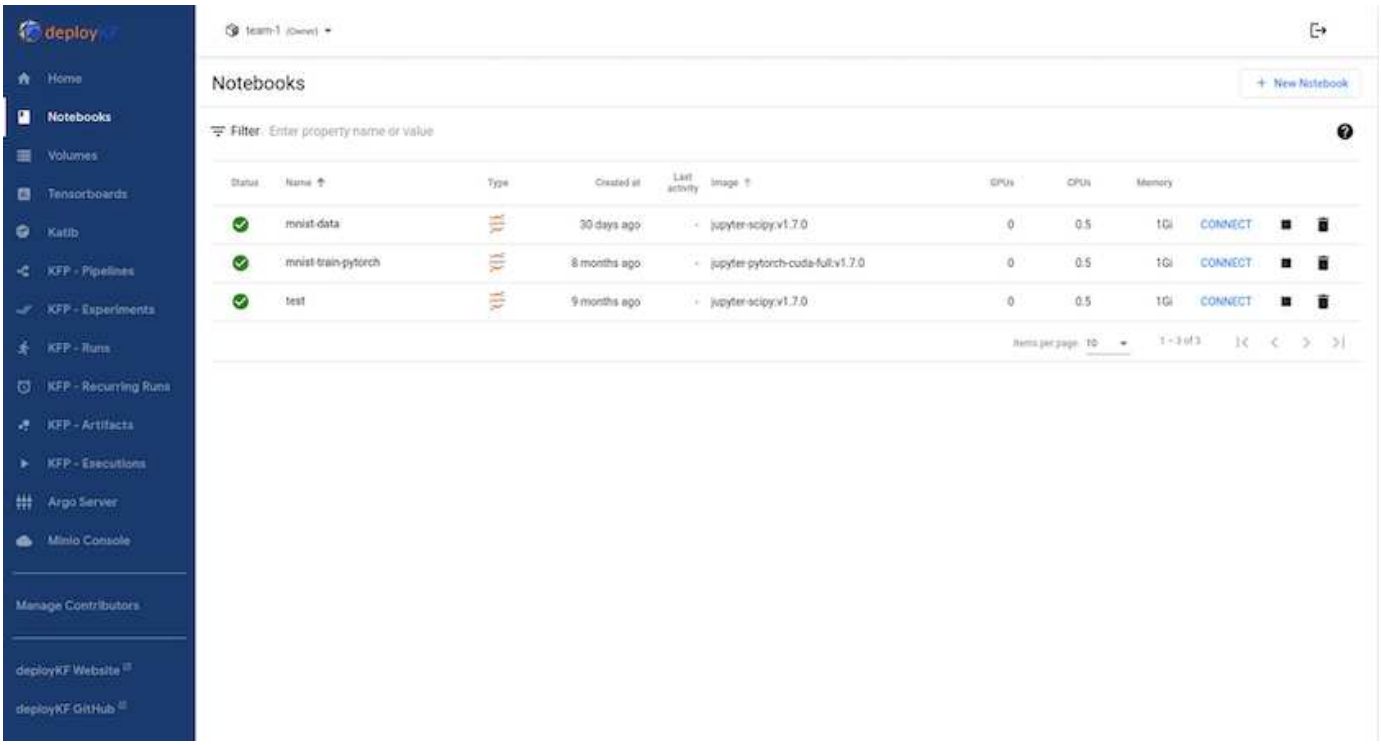
Kubeflow部署選項

部署 Kubeflow 有很多不同的選擇。請參閱"Kubeflow 官方文檔"取得部署選項列表，然後選擇最適合您需求的選項。

> 為了驗證目的，我們使用以下方式部署了 Kubeflow 1.7"部署KF" 0.1.1。

為資料科學家或開發人員提供 **Jupyter Notebook** 工作區

Kubeflow 能夠快速設定新的 Jupyter Notebook 伺服器作為資料科學家工作區。有關 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，請參閱 "Kubeflow 官方文檔"。



將NetApp DataOps 工具包與 Kubeflow 結合使用

這 "適用於 Kubernetes 的NetApp資料科學工具包"可以與Kubeflow結合使用。將NetApp資料科學工具包與 Kubeflow 結合使用可帶來以下好處：

- 資料科學家可以直接在 Jupyter Notebook 中執行高階NetApp資料管理操作，例如建立快照和複製。
- 可以使用 Kubeflow Pipelines 框架將進階NetApp資料管理作業（例如建立快照和複製）納入自動化工作流程。

請參閱 "Kubeflow 範例"有關將該工具包與 Kubeflow 一起使用的詳細信息，請參閱NetApp資料科學工具包 GitHub 儲存庫中的部分。

範例工作流程 - 使用 Kubeflow 和NetApp DataOps 工具包訓練影像辨識模型

本節介紹使用 Kubeflow 和NetApp DataOps Toolkit 訓練和部署用於影像辨識的神經網路的步驟。這旨在作為範例來展示結合NetApp儲存的訓練作業。

先決條件

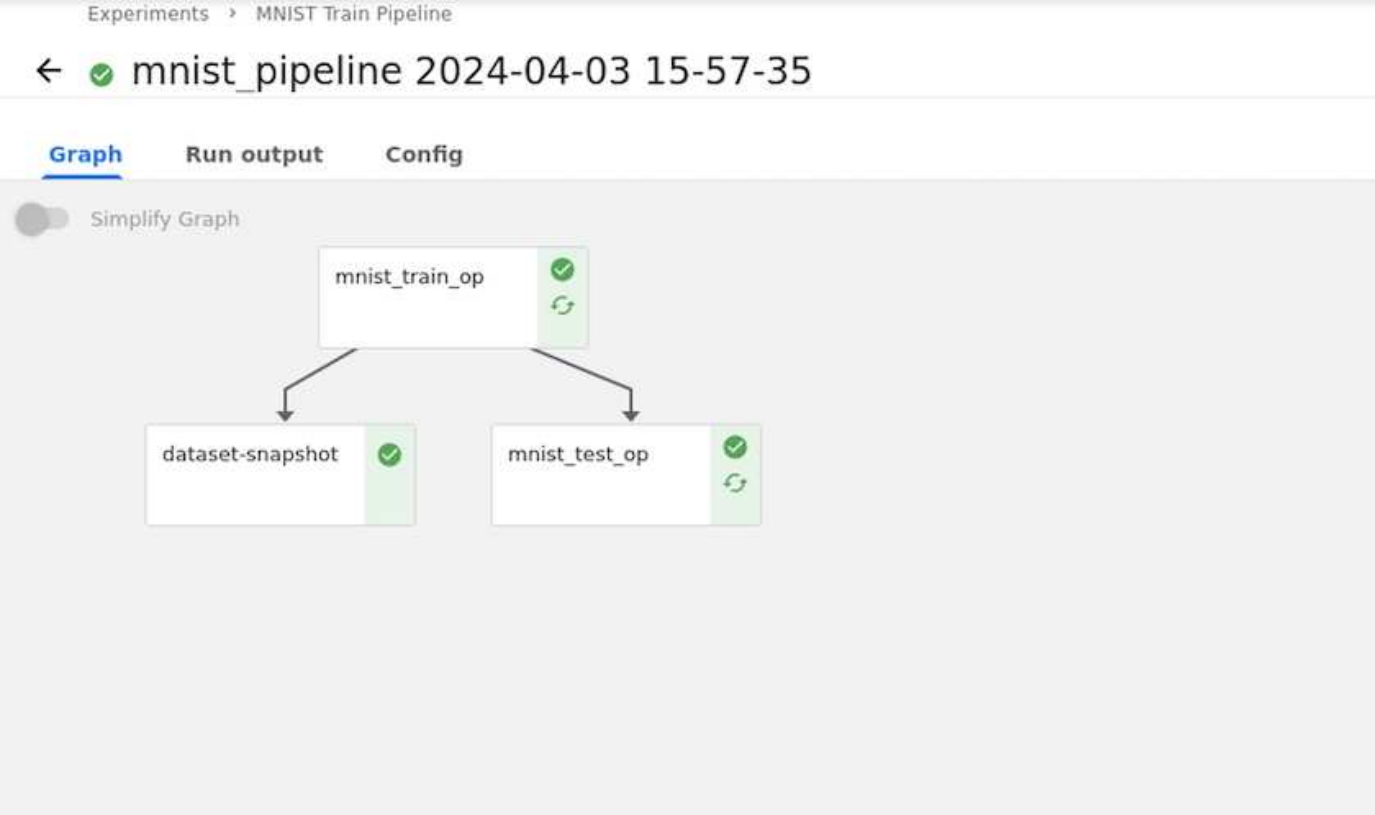
建立一個包含所需配置的 Dockerfile，用於 Kubeflow 管道內的訓練和測試步驟。以下是 Dockerfile 的範例 -

```
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"]
```

根據您的要求，安裝運行程式所需的所有必需程式庫和套件。在訓練機器學習模型之前，假設您已經有一個可執行的 Kubeflow 部署。

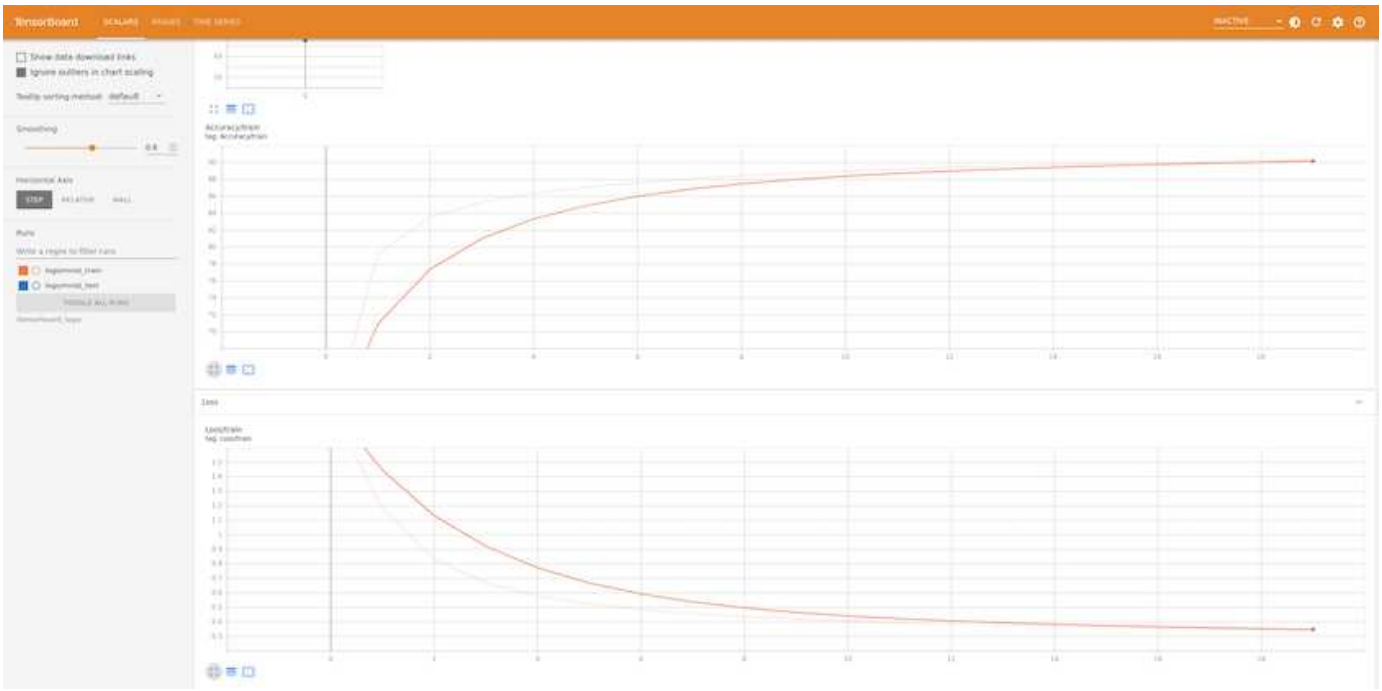
使用 **PyTorch** 和 **Kubeflow Pipelines** 在 **MNIST** 資料上訓練小型 **NN**

我們使用在 MNIST 資料上訓練的小型神經網路作為範例。 MNIST 資料集由 0-9 的手寫數位影像組成。影像尺寸為 28x28 像素。此資料集分為 60,000 張訓練影像和 10,000 張驗證影像。本實驗所採用的神經網路是一個2層前饋網路。訓練是使用 Kubeflow Pipelines 執行的。請參閱文檔 "這裡"了解更多。我們的 Kubeflow 管道包含了先決條件部分的 docker 映像。



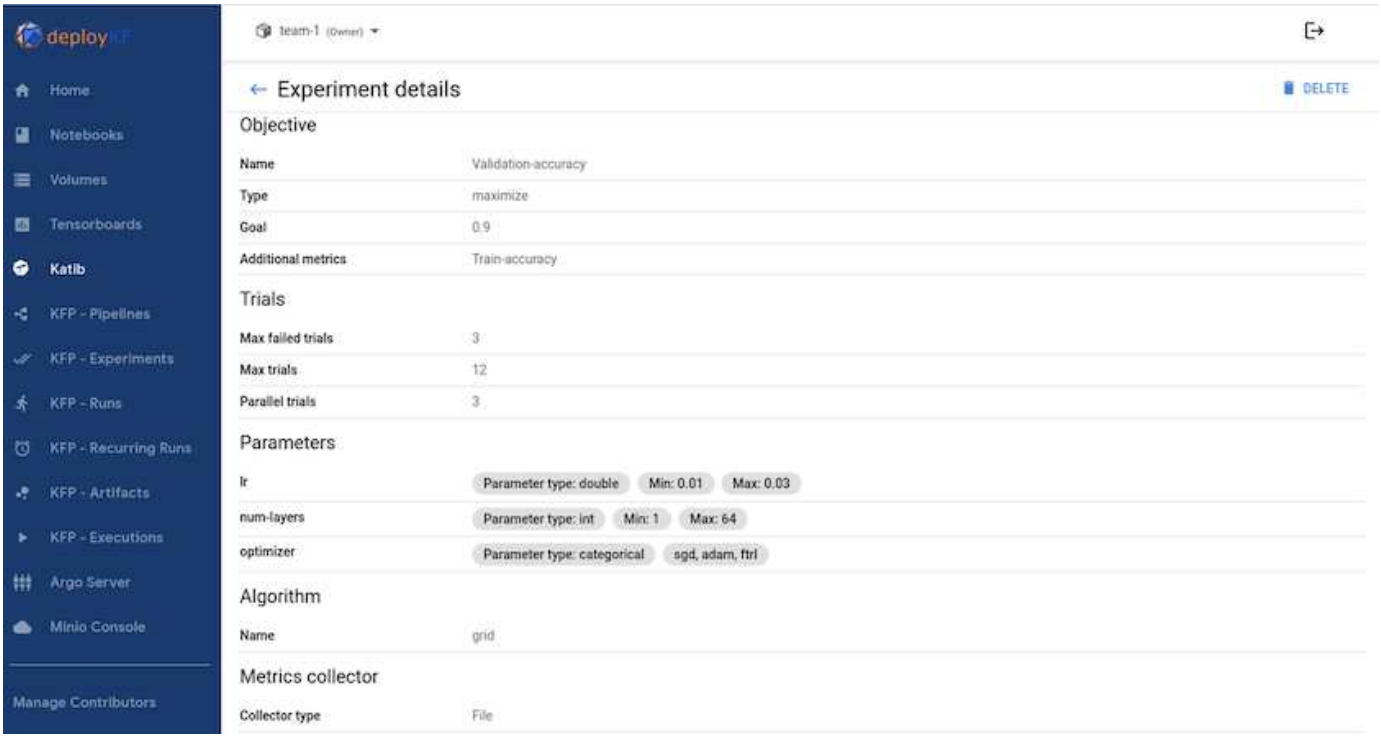
使用 **Tensorboard** 可視化結果

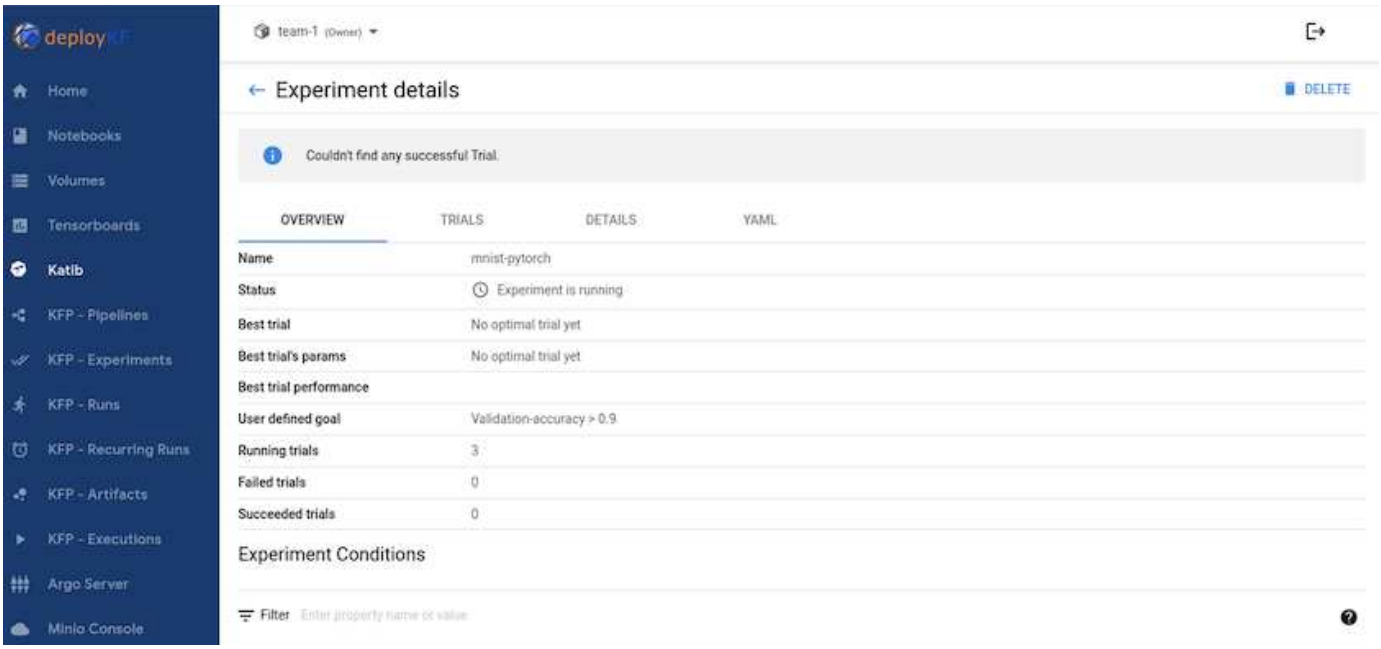
一旦模型訓練完成，我們就可以使用 Tensorboard 將結果視覺化。 "Tensorboard"作為 Kubeflow 儀表板上的功能提供。您可以為您的工作建立自訂張量板。以下的範例展示了訓練準確度與時期數以及訓練損失與時期數的關係圖。



使用 **Katib** 進行超參數實驗

"卡提布"是 Kubeflow 中的一個工具，可用來試驗模型超參數。要建立實驗，首先要定義所需的指標/目標。這通常是測試準確度。一旦定義了指標，選擇您想要使用的超參數（優化器/學習率/層數）。 Katib 使用使用者定義的值進行超參數掃描，以找到滿足所需指標的最佳參數組合。您可以在 UI 的每個部分中定義這些參數。或者，您可以定義一個具有必要規範的 **YAML** 檔案。以下是 Katib 實驗的說明 -





使用**NetApp**快照保存資料以實現可追溯性

在模型訓練期間，我們可能希望保存訓練資料集的快照以便於追溯。為此，我們可以向管道新增快照步驟，如下所示。要建立快照，我們可以使用 "適用於 Kubernetes 的NetApp DataOps 工具包"。

```
@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)
```

請參閱 "適用於 Kubeflow 的NetApp DataOps Toolkit 範例"了解更多。

Trident操作範例

本節包含您可能想要使用Trident執行的各種操作的範例。

導入現有磁碟區

如果您的NetApp儲存系統/平台上存在現有磁碟區，並且您想要將其安裝在 Kubernetes 叢集內的容器上，但這些磁碟區未與叢集中的 PVC 綁定，則必須匯入這些磁碟區。您可以使用Trident磁碟區匯入功能來匯入這些磁碟區。

以下範例指令顯示導入名為 `pb_fg_all`。有關 PVC 的更多信息，請參閱 "Kubernetes 官方文檔"。有關卷宗導入功能的更多信息，請參閱 "Trident文檔"。

一個 `accessModes`的價值 `ReadOnlyMany`在範例 PVC 規格檔中指定。有關 `accessMode`字段，請參閱 "Kubernetes 官方文檔"。

```
$ 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
```

提供新卷

您可以使用Trident在NetApp儲存系統或平台上設定新磁碟區。

使用 **kubectl** 設定新卷

以下範例指令顯示使用 kubectl 設定新的FlexVol volume。

一個 `accessModes`的價值 `ReadWriteMany`在下面的範例 PVC 定義檔中指定。有關 `accessMode`字段，請參閱 "Kubernetes 官方文檔"。

```
$ 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
```

使用**NetApp DataOps** 工具包配置新磁碟區

您也可以使用NetApp DataOps Toolkit for Kubernetes 在NetApp儲存系統或平台上設定新磁碟區。 NetApp DataOps Toolkit for Kubernetes 利用Trident來設定磁碟區，但簡化了使用者的流程。請參閱"文件"了解詳情。

AIPod部署的高效能作業範例

執行單節點 AI 工作負載

若要在 Kubernetes 叢集中執行單節點 AI 和 ML 作業，請從部署跳轉主機執行下列任務。使用Trident，您可以快速輕鬆地建立可能包含 PB 級資料的資料卷，以供 Kubernetes 工作負載存取。為了讓此類資料卷可從 Kubernetes pod 內部訪問，只需在 pod 定義中指定 PVC。

本節假設您已經將嘗試在 Kubernetes 叢集中執行的特定 AI 和 ML 工作負載容器化（以 Docker 容器格式）。

1. 以下範例指令展示如何為使用 ImageNet 資料集的 TensorFlow 基準工作負載建立 Kubernetes 作業。有關 ImageNet 資料集的更多信息，請參閱 "ImageNet 網站"。

   此範例作業請求八個 GPU，因此可以在具有八個或更多 GPU 的單一 GPU 工作節點上執行。此範例作業可以在叢集中提交，該叢集中不存在具有八個或更多 GPU 的工作節點，或目前正被另一個工作負載佔用。如果是，那麼作業將保持待處理狀態，直到有這樣的工作節點可用。

   此外，為了最大限度地提高儲存頻寬，包含所需訓練資料的磁碟區在該作業建立的 pod 中被安裝了兩次。另一個卷也安裝在 pod 中。第二卷將用於儲存結果和指標。這些磁碟區在作業定義中透過使用 PVC 的名稱來引用。有關 Kubernetes 作業的更多信息，請參閱 "Kubernetes 官方文檔"。

   一個 `emptyDir`音量 `medium`的價值 `Memory`安裝到 `/dev/shm`在此範例作業所建立的 pod 中。預設大小 `/dev/shm`Docker 容器運行時自動建立的虛擬磁碟區有時無法滿足 TensorFlow 的需求。安裝 `emptyDir`如下例所示，音量提供了足夠大的 `/dev/shm`虛擬卷。有關更多信息 `emptyDir`卷，參見 "Kubernetes 官方文檔"。

   此範例作業定義中指定的單一容器被賦予 `securityContext > privileged`的價值 `true`。該值意味著容器實際上在主機上具有 root 存取權限。在這種情況下使用此註釋，因為正在執行的特定工作負載需要 root 存取權。具體來說，工作負載執行的清除快取操作需要 root 存取權限。不管這是否 `privileged: true`註釋是否必要取決於您正在執行的特定工作負載的要求。

```
$ 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. 確認您在步驟 1 中建立的作業正在正確執行。下列範例命令確認已為該作業建立了一個 pod（如作業定義中所指定），並且該 pod 目前正在其中一個 GPU 工作節點上執行。

```
$ 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. 確認您在步驟 1 中建立的作業已成功完成。以下範例指令確認作業已成功完成。

```
$ 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. *可選：*清理工作成果。以下範例指令顯示刪除在步驟 1 中建立的作業物件。

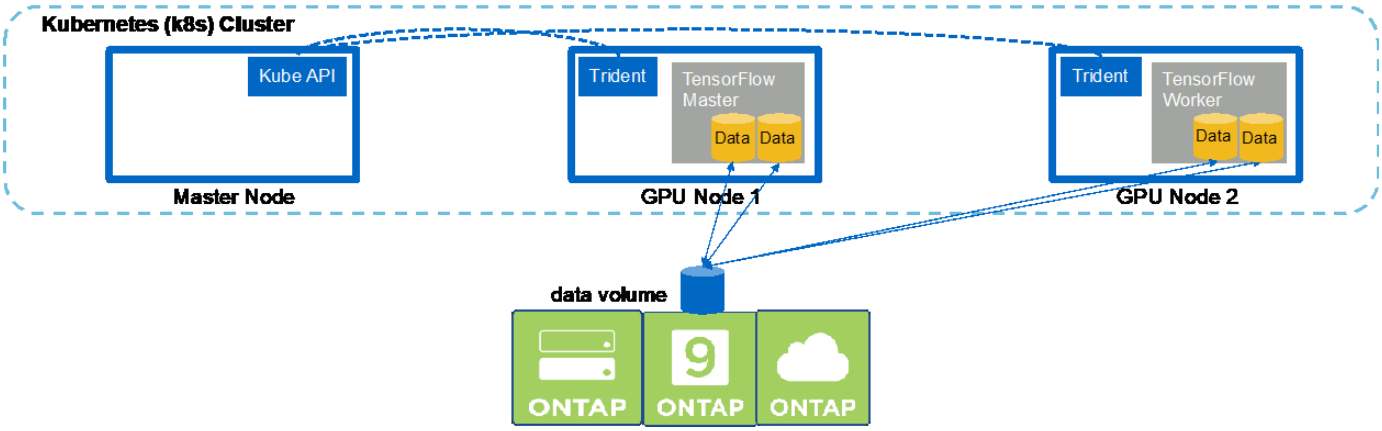
   當您刪除作業物件時，Kubernetes 會自動刪除任何關聯的 pod。

```
$ 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.
```

執行同步分散式 AI 工作負載

若要在 Kubernetes 叢集中執行同步多節點 AI 和 ML 作業，請在部署跳轉主機上執行下列任務。此過程使您能夠利用儲存在NetApp磁碟區上的數據，並使用比單一工作節點所能提供的更多的 GPU。請參考下圖以了解同步分散式 AI 作業的描述。

與非同步分散式作業相比，同步分散式作業可以幫助提高效能和訓練準確性。關於同步作業與非同步作業的優缺點的討論超出了本文檔的範圍。



1. 以下範例指令顯示如何建立一個工作器，該工作器參與本節範例中在單一節點上執行的相同 TensorFlow 基準測試作業的同步分散式執行"執行單節點 AI 工作負載"。在這個特定的例子中，只部署了一個工作器，因為作業是在兩個工作器節點上執行的。

   此範例工作器部署請求八個 GPU，因此可以在具有八個或更多 GPU 的單一 GPU 工作器節點上執行。如果您的 GPU 工作節點具有超過 8 個 GPU，為了最大限度地提高效能，您可能需要將此數字增加到等於您的工作節點所具有的 GPU 數量。有關 Kubernetes 部署的更多信息，請參閱 "Kubernetes 官方文檔"。

在此範例中建立了 Kubernetes 部署，因為這個特定的容器化工作程式永遠無法自行完成。因此，使用 Kubernetes 作業建構來部署它是沒有意義的。如果您的工作者被設計或編寫為自行完成，那麼使用作業建構來部署您的工作者可能是有意義的。

此範例部署規範中指定的 pod 被賦予 `hostNetwork`的價值 `true`。此值表示 pod 使用主機工作節點的網路堆疊，而不是 Kubernetes 通常為每個 pod 建立的虛擬網路堆疊。在這種情況下使用此註釋，因為特定的工作負載依賴 Open MPI、NCCL 和 Horovod 以同步分散式方式執行工作負載。因此，它需要存取主機網路堆疊。有關 Open MPI、NCCL 和 Horovod 的討論超出了本文檔的範圍。不管這是否 `hostNetwork: true` 註釋是否必要取決於您正在執行的特定工作負載的要求。有關 `hostNetwork`字段，請參閱 "Kubernetes 官方文檔"。

```
$ 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. 確認您在步驟 1 中建立的工作程序部署已成功啟動。以下範例命令確認已為部署建立了一個工作程序 pod（如部署定義所示），並且該 pod 目前正在其中一個 GPU 工作程序節點上執行。

```
$ 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. 為主伺服器建立一個 Kubernetes 作業，該主伺服器啟動、參與並追蹤同步多節點作業的執行。以下範例指令建立一個主伺服器，該主伺服器啟動、參與並追蹤在本節範例中在單一節點上執行的相同 TensorFlow 基準測試作業的同步分散式執行"執行單節點 AI 工作負載"。

   此範例主作業請求八個 GPU，因此可以在具有八個或更多 GPU 的單一 GPU 工作節點上執行。如果您的 GPU 工作節點具有超過 8 個 GPU，為了最大限度地提高效能，您可能需要將此數字增加到等於您的工作節點所具有的 GPU 數量。

   此範例作業定義中指定的主 Pod 被賦予 `hostNetwork`的價值 `true`就像工作艙被賦予了 `hostNetwork`的價值 `true`在步驟 1 中。有關為什麼需要此值的詳細信息，請參閱步驟 1。

```
$ 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. 確認您在步驟 3 中建立的主作業正在正確執行。下列範例指令確認已為該作業建立了一個主 pod（如作業定義所示），而該 pod 目前正在其中一個 GPU 工作節點上執行。您還應該看到，您在步驟 1 中最初看到的工作 pod 仍在運行，並且主 pod 和工作 pod 在不同的節點上運行。

```
$ 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. 確認您在步驟 3 中建立的主作業已成功完成。以下範例指令確認作業已成功完成。

```
$ 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. 當您不再需要工作部署時，請刪除它。以下範例指令顯示刪除在步驟 1 中建立的工作程序部署物件。

   當您刪除工作部署物件時，Kubernetes 會自動刪除任何關聯的工作容器。

```
$ 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. *可選：*清理主作業工件。以下範例指令顯示刪除在步驟 3 中建立的主作業物件。

   當您刪除主作業物件時，Kubernetes 會自動刪除任何關聯的主 pod。

```
$ 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.
```