NetApp的开源 MLOps

NetApp artificial intelligence solutions

NetApp
December 04, 2025

目录

NetApp的开源 MLOps

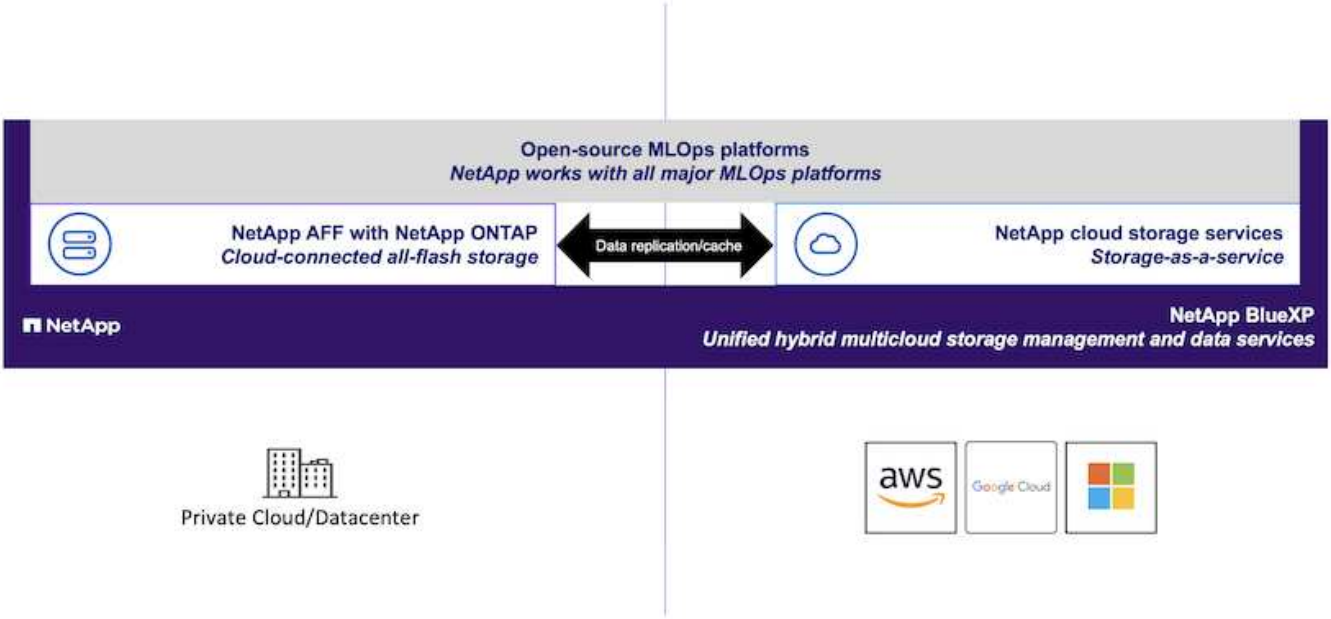
NetApp的开源 MLOps

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

各种规模和各个行业的公司和组织都在转向人工智能 (AI) 来解决现实世界的问题、提供创新的产品和服务，并在竞争日益激烈的市场中占据优势。许多组织正在转向开源 MLOps 工具，以跟上行业快速创新的步伐。这些开源工具提供了先进的功能和尖端的特性，但通常不考虑数据可用性和数据安全性。不幸的是，这意味着高技能的数据科学家被迫花费大量时间等待获取数据或等待基本的数据相关操作完成。通过将流行的开源 MLOps 工具与NetApp的智能数据基础架构相结合，组织可以加速其数据管道，从而加速其 AI 计划。他们可以从数据中释放价值，同时确保数据受到保护且安全。该解决方案展示了NetApp数据管理功能与几种流行的开源工具和框架的配对，以应对这些挑战。

以下列表重点介绍了此解决方案支持的一些关键功能：

- 用户可以快速配置由高性能、横向扩展NetApp存储支持的新的高容量数据卷和开发工作区。
- 用户可以几乎即时地克隆大容量数据卷和开发工作区，以便进行实验或快速迭代。
- 用户可以几乎即时保存大容量数据卷和开发工作区的快照，以进行备份和/或可追溯性/基准测试。



典型的 MLOps 工作流程包含开发工作区，通常采用以下形式"Jupyter 笔记本"；实验跟踪；自动化训练管道；数据管道；以及推理/部署。该解决方案重点介绍了几种不同的工具和框架，它们可以独立使用或结合使用来解决工作流程的不同方面。我们还展示了NetApp数据管理功能与每种工具的配对。该解决方案旨在提供构建模块，组织可据此构建针对其用例和要求的定制 MLOps 工作流程。

此解决方案涵盖以下工具/框架：

- "Apache Airflow"

- "JupyterHub"
- "Kubeflow"
- "机器学习流"

以下列表描述了独立或联合部署这些工具的常见模式。

- 联合部署 JupyterHub、MLflow 和 Apache Airflow - JupyterHub"Jupyter 笔记本" 、用于实验跟踪的 MLflow 以及用于自动化训练和数据管道的 Apache Airflow。
- 联合部署 Kubeflow 和 Apache Airflow - Kubeflow"Jupyter 笔记本" 、实验跟踪、自动化训练管道和推理；以及用于数据管道的 Apache Airflow。
- 将 Kubeflow 部署为一体化 MLOps 平台解决方案"Jupyter 笔记本"、实验跟踪、自动化训练和数据管道以及推理。

技术概述

本节重点介绍NetApp的 OpenSource MLOps 技术概述。

人工智能

人工智能是一门计算机科学学科，其中计算机经过训练可以模仿人类思维的认知功能。人工智能开发人员训练计算机以类似于人类甚至优于人类的方式学习和解决问题。深度学习和机器学习是人工智能的子领域。越来越多的组织采用 AI、ML 和 DL 来支持其关键业务需求。以下是一些示例：

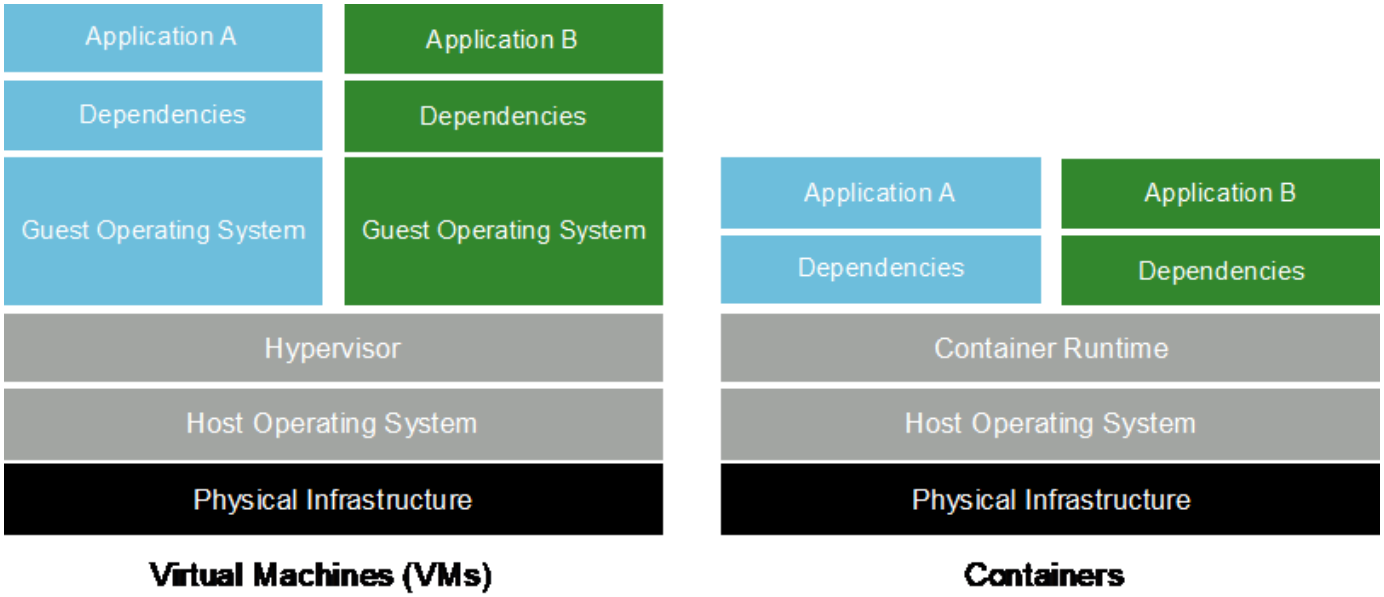
- 分析大量数据以发掘以前未知的商业见解
- 使用自然语言处理直接与客户互动
- 自动化各种业务流程和功能

现代人工智能训练和推理工作负载需要大规模并行计算能力。因此，GPU 越来越多地被用于执行 AI 操作，因为 GPU 的并行处理能力远远优于通用 CPU。

容器

容器是在共享主机操作系统内核上运行的隔离的用户空间实例。容器的采用正在迅速增加。容器提供许多与虚拟机 (VM) 相同的应用程序沙盒优势。然而，由于虚拟机所依赖的虚拟机管理程序和客户操作系统层已被消除，因此容器变得更加轻量级。下图描述了虚拟机与容器的可视化。

容器还允许直接将应用程序依赖项、运行时间等与应用程序高效地打包在一起。最常用的容器打包格式是Docker容器。以 Docker 容器格式容器化的应用程序可以在任何可以运行 Docker 容器的机器上执行。即使应用程序的依赖项不存在于机器上，情况也是如此，因为所有依赖项都打包在容器本身中。欲了解更多信息，请访问 "Docker 网站"。



Kubernetes

Kubernetes 是一个开源的、分布式的容器编排平台，最初由 Google 设计，现在由云原生计算基金会 (CNCF) 维护。 Kubernetes 支持容器化应用程序的部署、管理和扩展功能的自动化。近年来，Kubernetes 已经成为主流的容器编排平台。欲了解更多信息，请访问 "Kubernetes 网站"。

NetApp Trident

"Trident"支持在所有流行的NetApp存储平台上（公共云或内部）使用和管理存储资源，包括ONTAP （AFF、FAS、Select、Cloud、 Amazon FSx ONTAP）、 Azure NetApp Files服务和Google Cloud NetApp Volumes。Trident是一个符合容器存储接口 (CSI) 的动态存储编排器，可与 Kubernetes 原生集成。

NetApp DataOps 工具包

这"NetApp DataOps 工具包"是一个基于 Python 的工具，可简化由高性能、横向扩展NetApp存储支持的开发/培训工作区和推理服务器的管理。主要功能包括：

- 快速配置由高性能、横向扩展NetApp存储支持的全新高容量工作区。
- 近乎即时地克隆高容量工作区，以实现实验或快速迭代。
- 近乎即时地保存高容量工作区的快照，以用于备份和/或可追溯性/基准测试。
- 近乎即时地提供、克隆和快照大容量、高性能数据卷。

Apache Airflow

Apache Airflow 是一个开源工作流管理平台，支持以编程方式编写、调度和监控复杂的企业工作流。它通常用于自动化 ETL 和数据管道工作流程，但并不局限于这些类型的工作流程。 Airflow 项目由 Airbnb 发起，但后来在业界变得非常流行，现在由 Apache 软件基金会赞助。 Airflow 是用 Python 编写的，Airflow 工作流是通过 Python 脚本创建的，并且 Airflow 是在“配置即代码”的原则下设计的。许多企业 Airflow 用户现在在 Kubernetes 上运行 Airflow。

有向无环图（**DAG**）

在 Airflow 中，工作流被称为有向无环图 (DAG)。 DAG 由按顺序、并行或两者结合执行的任务组成，具体取决于 DAG 定义。 Airflow 调度程序在一组工作器上执行各个任务，遵守 DAG 定义中指定的任务级依赖关系。 DAG 是通过 Python 脚本定义和创建的。

Jupyter 笔记本

Jupyter Notebooks 是类似 wiki 的文档，包含实时代码和描述性文本。 Jupyter Notebooks 在 AI 和 ML 社区中被广泛用作记录、存储和共享 AI 和 ML 项目的一种方式。有关 Jupyter Notebooks 的更多信息，请访问 "Jupyter 网站"。

Jupyter Notebook 服务器

Jupyter Notebook 服务器是一个开源 Web 应用程序，允许用户创建 Jupyter Notebook。

JupyterHub

JupyterHub 是一个多用户应用程序，允许个人用户配置和访问他们自己的 Jupyter Notebook 服务器。有关 JupyterHub 的更多信息，请访问 "JupyterHub 网站"。

机器学习流

MLflow 是一个流行的开源 AI 生命周期管理平台。 MLflow 的主要功能包括 AI/ML 实验跟踪和 AI/ML 模型库。有关 MLflow 的更多信息，请访问 "MLflow 网站"。

Kubeflow

Kubeflow 是 Kubernetes 的开源 AI 和 ML 工具包，最初由 Google 开发。 Kubeflow 项目使 Kubernetes 上 AI 和 ML 工作流的部署变得简单、可移植且可扩展。 Kubeflow 抽象了 Kubernetes 的复杂性，使数据科学家能够专注于他们最了解的领域——数据科学。请参见下图以了解可视化效果。对于喜欢一体化 MLOps 平台的组织来说，Kubeflow 是一个不错的开源选择。欲了解更多信息，请访问 "Kubeflow 网站"。

Kubeflow 管道

Kubeflow Pipelines 是 Kubeflow 的关键组件。 Kubeflow Pipelines 是一个用于定义和部署可移植、可扩展的 AI 和 ML 工作流的平台和标准。有关详细信息，请参阅 "Kubeflow 官方文档"。

Kubeflow 笔记本

Kubeflow 简化了 Kubernetes 上 Jupyter Notebook 服务器的配置和部署。有关 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，请参阅 "Kubeflow 官方文档"。

卡提布

Katib 是一个用于自动化机器学习 (AutoML) 的 Kubernetes 原生项目。 Katib 支持超参数调整、早期停止和神经架构搜索 (NAS)。 Katib 是一个与机器学习 (ML) 框架无关的项目。它可以调整用户选择的任何语言编写的应用程序的超参数，并且原生支持许多 ML 框架，例如 TensorFlow、MXNet、PyTorch、XGBoost 等。 Katib 支持许多不同的 AutoML 算法，例如贝叶斯优化、Parzen 估计器树、随机搜索、协方差矩阵自适应进化策略、超频、高效神经架构搜索、可微分架构搜索等等。有关 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，请参阅 "Kubeflow 官方文档"。

NetApp ONTAP

ONTAP 9 是NetApp最新一代存储管理软件，它支持企业实现基础架构现代化并过渡到云就绪数据中心。ONTAP利用业界领先的数据管理功能，只需一套工具即可管理和保护数据，无论数据位于何处。您还可以将数据自由移动到任何需要的地方：边缘、核心或云端。 ONTAP 9 包含众多功能，可简化数据管理、加速和保护关键数据，并支持跨混合云架构的下一代基础架构功能。

简化数据管理

数据管理对于企业 IT 运营和数据科学家至关重要，以便将适当的资源用于 AI 应用程序和训练 AI/ML 数据集。以下有关NetApp技术的附加信息超出了本次验证的范围，但可能与您的部署相关。

ONTAP数据管理软件包括以下功能，可简化操作并降低总运营成本：

- 内联数据压缩和扩展重复数据删除。数据压缩减少了存储块内部浪费的空间，重复数据删除显著增加了有效容量。这适用于本地存储的数据和分层到云的数据。
- 最小、最大和自适应服务质量 (AQoS)。细粒度的服务质量 (QoS) 控制有助于维持高度共享环境中关键应用程序的性能水平。
- NetApp FabricPool。提供冷数据自动分层到公共和私有云存储选项，包括 Amazon Web Services (AWS)、Azure 和NetApp StorageGRID存储解决方案。有关FabricPool的更多信息，请参阅 "TR-4598：FabricPool最佳实践"。

加速并保护数据

ONTAP提供卓越级别的性能和数据保护，并通过以下方式扩展这些功能：

- 性能和更低的延迟。 ONTAP以尽可能低的延迟提供尽可能高的吞吐量。
- 数据保护。ONTAP提供内置数据保护功能，并在所有平台上提供通用管理。
- NetApp卷加密 (NVE)。 ONTAP提供原生卷级加密，同时支持板载和外部密钥管理。
- 多租户和多因素身份验证。 ONTAP支持以最高级别的安全性共享基础设施资源。

面向未来的基础设施

ONTAP具有以下功能，可帮助满足苛刻且不断变化的业务需求：

- 无缝扩展和无中断运行。 ONTAP支持无中断地向现有控制器和横向扩展集群添加容量。客户可以升级到最新技术，而无需昂贵的数据迁移或中断。
- 云连接。 ONTAP是与云连接最紧密的存储管理软件，在所有公共云中提供软件定义存储和云原生实例的选项。
- 与新兴应用程序的集成。 ONTAP使用支持现有企业应用的相同基础架构，为下一代平台和应用（如自动驾驶汽车、智能城市和工业 4.0）提供企业级数据服务。

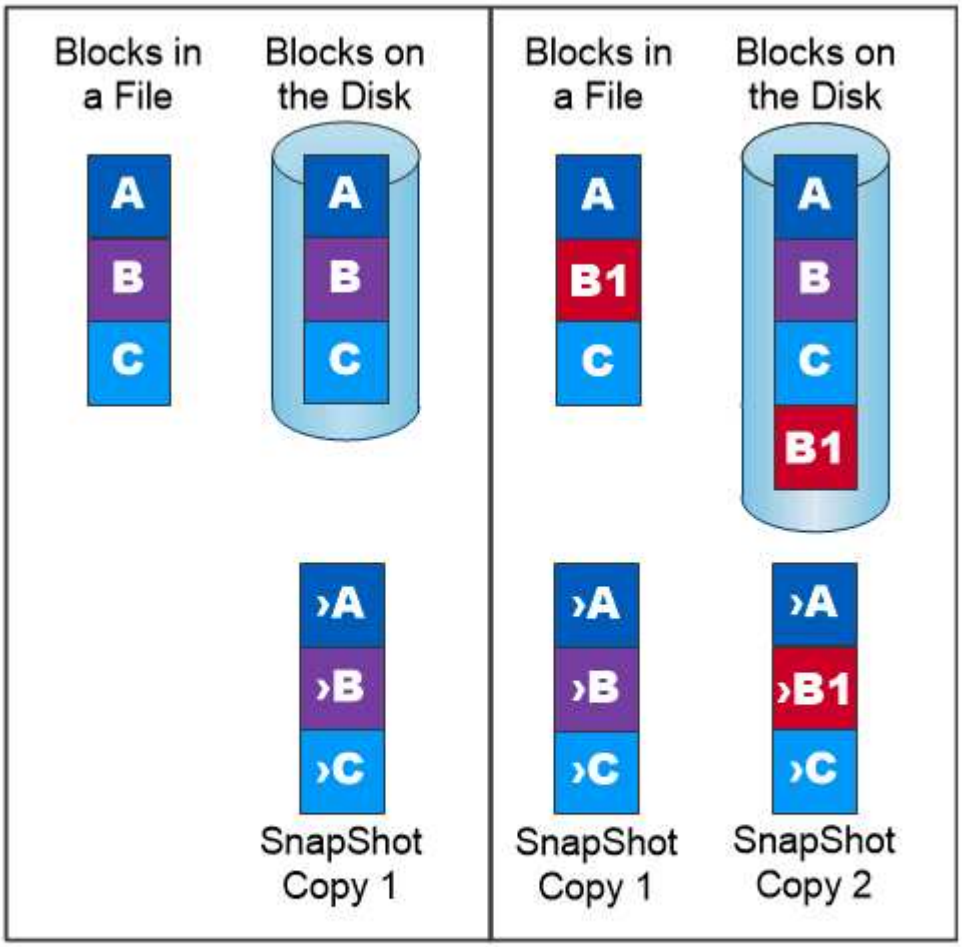
NetApp Snapshot 副本

NetApp Snapshot 副本是卷的只读、时间点映像。该图像占用的存储空间极小，并且产生的性能开销可以忽略不计，因为它仅记录自上次 Snapshot 副本创建以来对文件的更改，如下图所示。

Snapshot 副本的效率归功于核心ONTAP存储虚拟化技术，即任意位置写入文件布局 (WAFL)。与数据库一样，WAFL使用元数据指向磁盘上的实际数据块。但是，与数据库不同， WAFL不会覆盖现有块。它将更新的数据写

入新块并更改元数据。这是因为ONTAP在创建 Snapshot 副本时引用元数据，而不是复制数据块，所以 Snapshot 副本非常高效。这样做可以消除其他系统在定位要复制的块时产生的寻道时间，以及复制本身的成本。

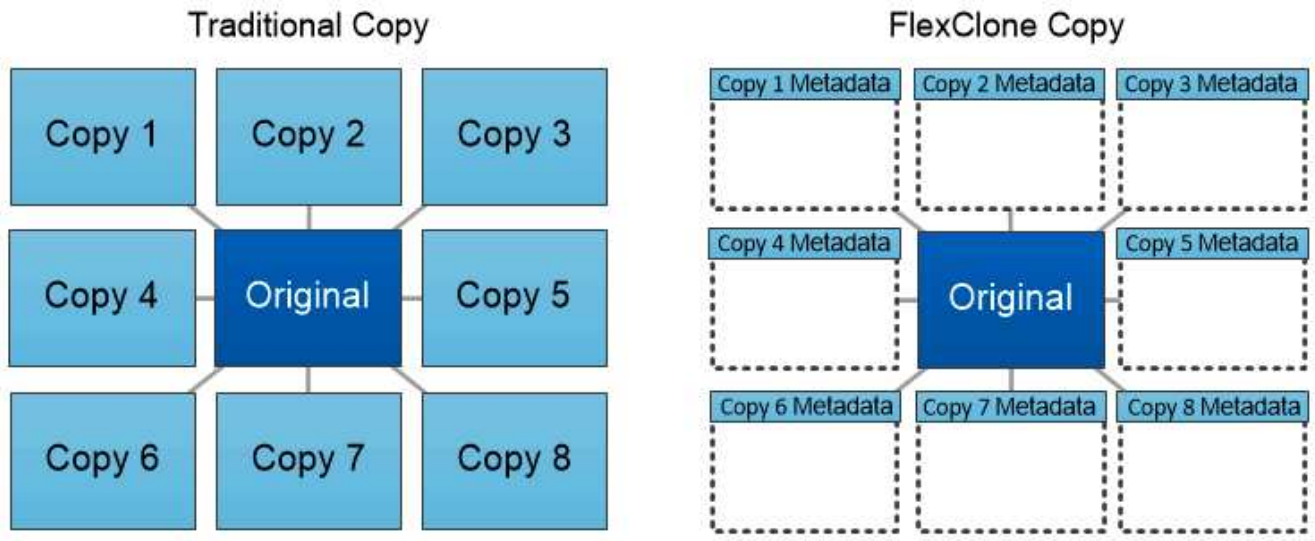
您可以使用 Snapshot 副本来恢复单个文件或 LUN，或者还原卷的全部内容。 ONTAP将 Snapshot 副本中的指针信息与磁盘上的数据进行比较，以重建丢失或损坏的对象，而无需停机或造成显著的性能成本。



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

NetApp FlexClone 技术

NetApp FlexClone技术参考 Snapshot 元数据来创建卷的可写时间点副本。副本与其父级共享数据块，除了元数据所需的存储空间外，不消耗任何存储空间，直到将更改写入副本为止，如下图所示。传统的复制可能需要几分钟甚至几小时才能创建，而FlexClone软件可以让您几乎立即复制最大的数据集。这使得它非常适合需要相同数据集的多个副本（例如，开发工作区）或数据集的临时副本（针对生产数据集测试应用程序）的情况。



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

NetApp SnapMirror数据复制技术

NetApp SnapMirror软件是一种跨数据结构的经济高效、易于使用的统一复制解决方案。它通过 LAN 或 WAN 高速复制数据。它为所有类型的应用程序（包括虚拟和传统环境中的关键业务应用程序）提供高数据可用性和快速数据复制。当您将数据复制到一个或多个NetApp存储系统并不断更新辅助数据时，您的数据将保持最新状态并可随时使用。不需要外部复制服务器。下图是利用SnapMirror技术的架构示例。

SnapMirror软件通过网络仅发送更改的块来利用NetApp ONTAP存储效率。 SnapMirror软件还使用内置网络压缩来加速数据传输并将网络带宽利用率降低高达 70%。借助SnapMirror技术，您可以利用一个精简复制数据流来创建一个存储库，该存储库同时维护活动镜像和之前的时间点副本，从而将网络流量减少高达 50%。

NetApp BlueXP复制和同步

"BlueXP复制和同步"是NetApp 的一项快速、安全的数据同步服务。无论您需要在本地 NFS 或 SMB 文件共享、NetApp StorageGRID、 NetApp ONTAP S3、 Google Cloud NetApp Volumes、 Azure NetApp Files、AWS S3、AWS EFS、Azure Blob、Google Cloud Storage 还是 IBM Cloud Object Storage 之间传输文件， BlueXP Copy and Sync 都能快速安全地将文件移动到您需要的位置。

数据传输完成后，可在源端和目标端完全使用。 BlueXP Copy and Sync 可以在触发更新时按需同步数据，或者根据预定义的时间表连续同步数据。无论如何， BlueXP Copy and Sync 仅移动增量，因此在数据复制上花费的时间和金钱被最小化。

BlueXP Copy and Sync 是一种软件即服务 (SaaS) 工具，其设置和使用极其简单。 BlueXP Copy 和 Sync 触发的数据传输由数据代理执行。 BlueXP Copy 和 Sync 数据代理可以部署在 AWS、Azure、Google Cloud Platform 或本地。

NetApp XCP

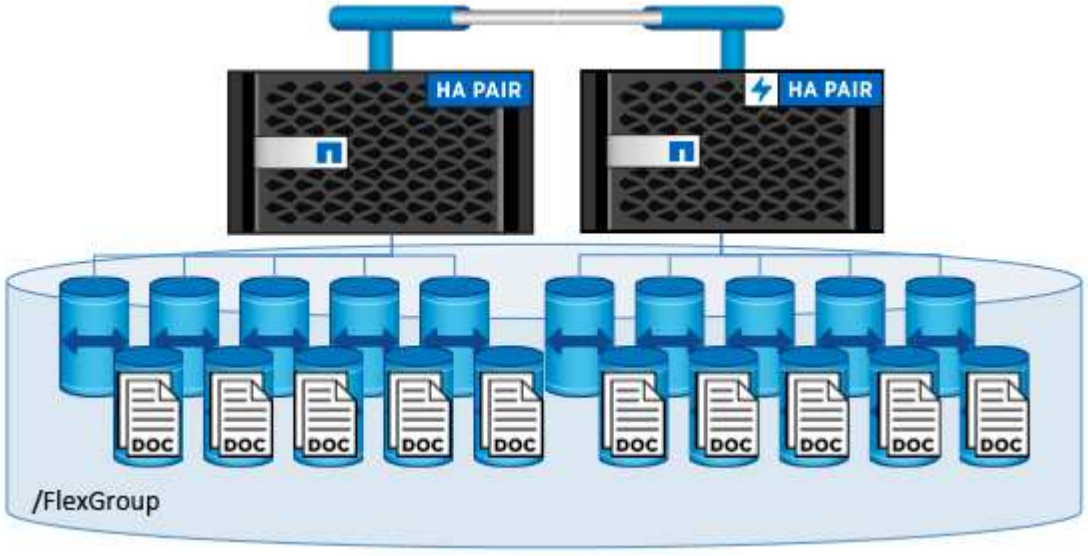
"NetApp XCP"是一款基于客户端的软件，用于任意到NetApp和NetApp到NetApp 的数据迁移和文件系统洞察。XCP 旨在通过利用所有可用的系统资源来处理大容量数据集和高性能迁移，从而实现扩展并实现最大性能。XCP 可帮助您全面了解文件系统，并提供生成报告的选项。

NetApp ONTAP FlexGroup卷

训练数据集可能包含数十亿个文件。文件可以包括文本、音频、视频和其他形式的非结构化数据，这些数据必须存储和处理才能并行读取。存储系统必须存储大量小文件，并且必须并行读取这些文件以实现顺序和随机 I/O。

FlexGroup卷是一个由多个组成成员卷组成的单一命名空间，如下图所示。从存储管理员的角度来看，FlexGroup卷的管理和行为类似于NetApp FlexVol volume。 FlexGroup卷中的文件被分配给各个成员卷，并且不会跨卷或节点进行条带化。它们支持以下功能：

- FlexGroup卷为高元数据工作负载提供了数 PB 的容量和可预测的低延迟。
- 它们支持同一命名空间中最多 4000 亿个文件。
- 它们支持跨 CPU、节点、聚合体和组成FlexVol卷的 NAS 工作负载的并行操作。



架构

该解决方案不依赖于特定的硬件。该解决方案与NetApp Trident支持的任何NetApp物理存储设备、软件定义实例或云服务兼容。示例包括NetApp AFF存储系统、 Amazon FSx ONTAP、 Azure NetApp Files、 Google Cloud NetApp Volumes或NetApp Cloud Volumes ONTAP实例。此外，只要所使用的 Kubernetes 版本受到NetApp Trident和正在实施的其他解决方案组件的支持，该解决方案就可以在任何 Kubernetes 集群上实施。有关Trident支持的 Kubernetes 版本列表，请参阅 "Trident文档"。有关用于验证此解决方案的各个组件的环境的详细信息，请参阅下表。

Apache Airflow 验证环境

软件组件	版本
Apache Airflow	2.0.1，通过以下方式部署"Apache Airflow Helm 图表 "8.0.8
Kubernetes	1.18

软件组件	版本
NetApp Trident	21.01

JupyterHub 验证环境

软件组件	版本
JupyterHub	4.1.5，通过部署"JupyterHub Helm 图表"3.3.7
Kubernetes	1.29
NetApp Trident	24.02

MLflow 验证环境

软件组件	版本
机器学习流	2.14.1，通过部署"MLflow Helm 图表"1.4.12
Kubernetes	1.29
NetApp Trident	24.02

Kubeflow 验证环境

软件组件	版本
Kubeflow	1.7，通过部署"部署KF"0.1.1
Kubernetes	1.26
NetApp Trident	23.07

支持

NetApp不为 Apache Airflow、JupyterHub、MLflow、Kubeflow 或 Kubernetes 提供企业支持。如果您对完全支持的 MLOps 平台感兴趣，"联系NetApp"了解NetApp与合作伙伴联合提供的全面支持的 MLOps 解决方案。

NetApp Trident配置

NetApp AIPod部署的Trident后端示例

在使用Trident在 Kubernetes 集群中动态配置存储资源之前，您必须创建一个或多个Trident后端。以下示例代表了如果您要在以下位置部署此解决方案的组件，您可能需要创建的不同类型的后端："NetApp AIPod" 。有关后端的更多信息，以及其他平台/环境的后端示例，请参阅"Trident文档"。

1. NetApp建议为您的AIPod创建支持FlexGroup的Trident Backend。

   下面的示例命令展示了如何为AIPod存储虚拟机 (SVM) 创建支持FlexGroup的Trident Backend。此后端使用 `ontap-nas-flexgroup`存储驱动程序。 ONTAP支持两种主要数据卷类型： FlexVol和FlexGroup。 FlexVol卷

的大小受到限制（截至撰写本文时，最大大小取决于具体的部署）。另一方面， FlexGroup卷可以线性扩展到最多 20PB 和 4000 亿个文件，从而提供单一命名空间，大大简化数据管理。因此， FlexGroup卷最适合依赖大量数据的 AI 和 ML 工作负载。

如果您处理的数据量较小，并且想要使用FlexVol卷而不是FlexGroup卷，则可以创建使用 `ontap-nas`存储驱动程序，而不是 `ontap-nas-flexgroup`存储驱动程序。

```
$ 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还建议创建支持FlexVol的Trident Backend。您可能希望使用FlexVol卷来托管持久应用程序、存储结果、输出、调试信息等。如果要使用FlexVol卷，则必须创建一个或多个启用FlexVol的Trident后端。下面的示例命令显示如何创建启用单个FlexVol的Trident后端。

```
$ 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 存储类示例

在使用Trident在 Kubernetes 集群中动态配置存储资源之前，您必须创建一个或多个 Kubernetes StorageClasses。以下示例代表了如果您在以下位置部署此解决方案的组件，则可能需要创建的不同类型的 StorageClasses："NetApp AIPod" 。有关 StorageClasses 的更多信息，以及其他平台/环境的 StorageClasses 示例，请参阅"Trident文档"。

1. NetApp建议为您在本节中创建的支持FlexGroup的Trident Backend 创建 StorageClass"NetApp AIPod部署的Trident后端示例"中，步骤 1。下面的示例命令显示了如何创建多个 StorageClasses，这些 StorageClasses 与本节中创建的示例 Backend 相对应"NetApp AIPod部署的Trident后端示例"，步骤 1 - 利用"基于 RDMA 的 NFS"还有一个则不然。

   为了确保持久卷在删除相应的 PersistentVolumeClaim (PVC) 时不会被删除，以下示例使用 `reclaimPolicy`的价值 `Retain`。有关 `reclaimPolicy`字段，请参阅官方 "Kubernetes 文档"。

   注意：以下示例 StorageClasses 使用的最大传输大小为 262144。要使用此最大传输大小，您必须在ONTAP系统上相应地配置最大传输大小。请参阅"ONTAP 文档"了解详情。

   注：要使用 NFS over RDMA，您必须在ONTAP系统上配置 NFS over RDMA。请参阅"ONTAP 文档"了解详情。

   注意：以下示例中，StorageClass 定义文件中的 storagePool 字段指定了具体的 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还建议创建一个与您在本节中创建的支持FlexVol的Trident Backend 相对应的 StorageClass"用于AIPod部署的Trident后端示例"中，步骤 2。下面的示例命令显示如何为FlexVol卷创建单个StorageClass。

   注意：在下面的示例中，StorageClass 定义文件中的 storagePool 字段未指定特定的 Backend。当你使用 Kubernetes 来管理使用此 StorageClass 的卷时， Trident会尝试使用任何可用的后端，该后端使用 `ontap-nas`司机。

```
$ 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

本节介绍在 Kubernetes 集群中部署 Airflow 必须完成的任务。

> 可以在 Kubernetes 以外的平台上部署 Airflow。在 Kubernetes 以外的平台上部署 Airflow 超出了本解决方案的范围。

前提条件

在执行本节概述的部署练习之前，我们假设您已经执行了以下任务：

1. 您已经有一个可以运行的 Kubernetes 集群。
2. 您已经在 Kubernetes 集群中安装并配置了NetApp Trident 。有关Trident的更多详细信息，请参阅"Trident文档"。

安装 **Helm**

Airflow 使用 Helm（Kubernetes 的流行包管理器）进行部署。在部署 Airflow 之前，必须在部署跳转主机上安装 Helm。要在部署跳转主机上安装 Helm，请按照 "安装说明"在 Helm 官方文档中。

设置默认 **Kubernetes StorageClass**

在部署 Airflow 之前，您必须在 Kubernetes 集群中指定一个默认 StorageClass。 Airflow 部署过程尝试使用默认 StorageClass 来配置新的持久卷。如果没有指定 StorageClass 作为默认 StorageClass，则部署失败。要在集群中指定默认 StorageClass，请按照"Kubeflow部署"部分。如果您已经在集群中指定了默认 StorageClass，则可以跳过此步骤。

使用 **Helm** 部署 **Airflow**

要使用 Helm 在 Kubernetes 集群中部署 Airflow，请从部署跳转主机执行以下任务：

1. 按照以下说明使用 Helm 部署 Airflow "部署说明"查看 Artifact Hub 上的官方 Airflow 图表。下面的示例命令展示了使用 Helm 部署 Airflow。修改、添加和/或删除 `custom- values.yaml`根据您的环境和所需配置，根据需要创建文件。

```
$ 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
```

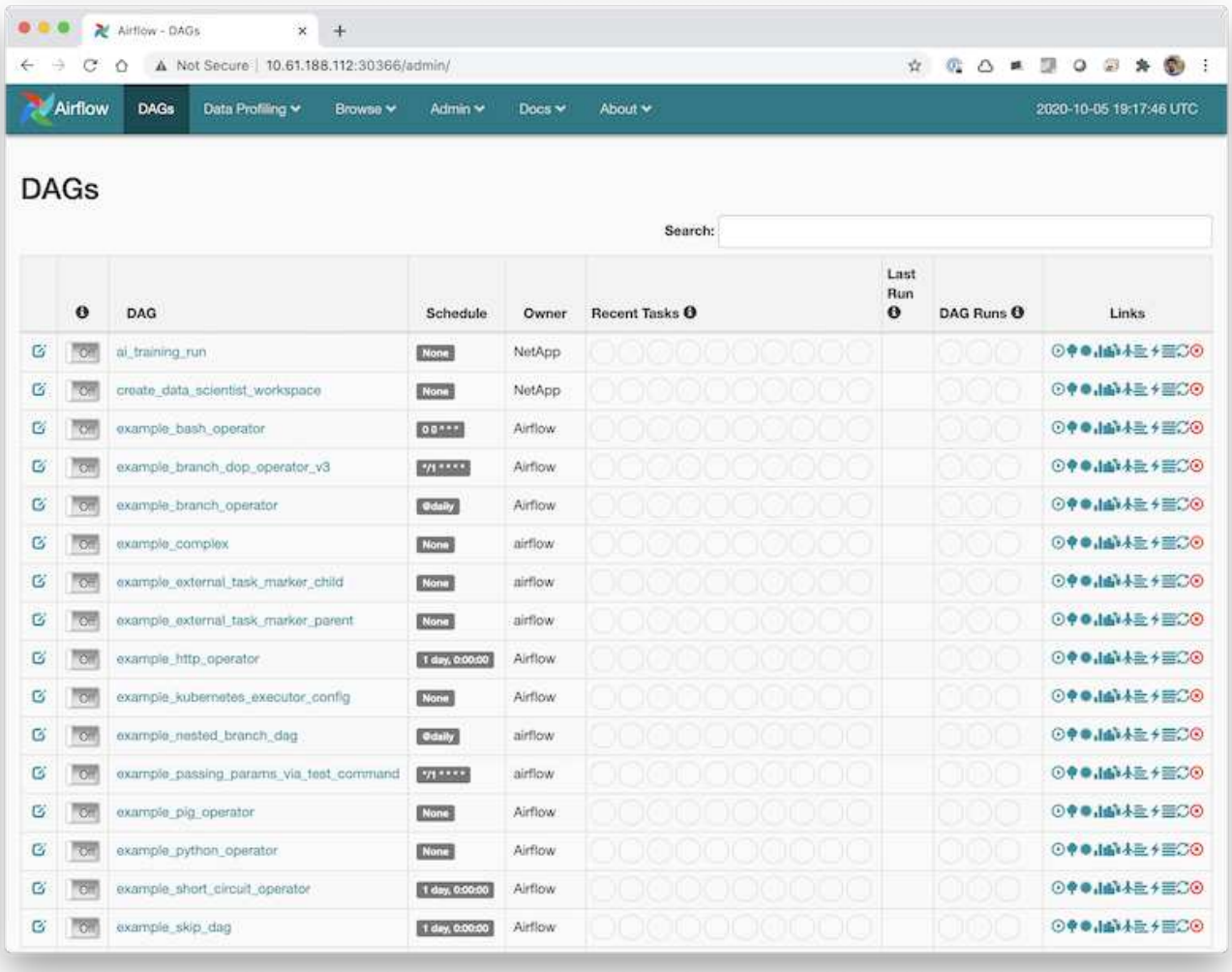
2. 确认所有 Airflow pod 均已启动并正在运行。所有 pod 启动可能需要几分钟时间。

```
$ 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. 按照步骤 1 中使用 Helm 部署 Airflow 时打印到控制台的说明获取 Airflow Web 服务 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. 确认您可以访问 Airflow Web 服务。



将NetApp DataOps 工具包与 Airflow 结合使用

这 "适用于 Kubernetes 的NetApp DataOps 工具包"可以与 Airflow 结合使用。将NetApp DataOps Toolkit 与 Airflow 结合使用，您可以将NetApp数据管理操作（例如创建快照和克隆）合并到由 Airflow 协调的自动化工作流程中。

请参阅 "气流示例"有关将该工具包与 Airflow 结合使用的详细信息，请参阅NetApp DataOps Toolkit GitHub 存储库中的部分。

JupyterHub

JupyterHub 部署

本节介绍在 Kubernetes 集群中部署 JupyterHub 必须完成的任务。

> 可以在 Kubernetes 以外的平台上部署 JupyterHub。在 Kubernetes 以外的平台上部署 JupyterHub 超出了本解决方案的范围。

前提条件

在执行本节概述的部署练习之前，我们假设您已经执行了以下任务：

1. 您已经有一个可以运行的 Kubernetes 集群。
2. 您已经在 Kubernetes 集群中安装并配置了NetApp Trident 。有关Trident的更多详细信息，请参阅"Trident文档"。

安装 **Helm**

JupyterHub 使用 Helm（Kubernetes 的流行包管理器）进行部署。在部署 JupyterHub 之前，您必须在 Kubernetes 控制节点上安装 Helm。要安装 Helm，请按照 "安装说明"在 Helm 官方文档中。

设置默认 **Kubernetes StorageClass**

在部署 JupyterHub 之前，您必须在 Kubernetes 集群中指定一个默认 StorageClass。要在集群中指定默认 StorageClass，请按照"Kubeflow部署"部分。如果您已经在集群中指定了默认 StorageClass，则可以跳过此步骤。

部署 **JupyterHub**

完成上述步骤后，您现在可以部署 JupyterHub。 JupyterHub 部署需要以下步骤：

配置 **JupyterHub** 部署

在部署之前，最好针对各自的环境优化 JupyterHub 部署。您可以创建一个 **config.yaml** 文件并在使用 Helm 图表部署期间使用它。

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

> 在此 config.yaml 文件中，您可以为NetApp Trident StorageClass 设置 **(singleuser.storage.dynamic.storageClass)** 参数。这是用于为各个用户工作区配置卷的存储类。

添加共享卷

如果您想为所有 JupyterHub 用户使用共享卷，您可以相应地调整您的 **config.yaml**。例如，如果您有一个名为 jupyterhub-shared-volume 的共享 PersistentVolumeClaim，则可以将其作为 /home/shared 挂载在所有用户 pod 中，如下所示：

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

这是可选步骤，您可以根据需要调整这些参数。

使用 **Helm Chart** 部署 **JupyterHub**

让 Helm 了解 JupyterHub Helm 图表存储库。

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

这应该显示如下输出：

```
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!□
```

现在通过从包含您的 config.yaml 的目录运行以下命令来安装由您的 config.yaml 配置的图表：

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

在此示例中：

<helm-release-name> 设置为 my-jupyterhub，这将是您的 JupyterHub 版本的名称。 <k8s-namespace> 设置为 my-namespace，这是您要安装 JupyterHub 的命名空间。如果命名空间不存在，则使用 --create-namespace 标志来创建命名空间。 --values 标志指定包含所需配置选项的 config.yaml 文件。

检查部署

在步骤 2 运行时，您可以通过以下命令看到正在创建的 pod：

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

等待 hub 和 proxy pod 进入 Running 状态。

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

访问 JupyterHub

找到我们可以用来访问 JupyterHub 的 IP。运行以下命令，直到代理公共服务的 EXTERNAL-IP 可用，如示例输出所示。

> 我们在 config.yaml 文件中使用了 NodePort 服务，您可以根据您的设置（例如 LoadBalancer）调整您的环境。

```
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
```

要使用 JupyterHub，请在浏览器中输入代理公共服务的外部 IP。

将NetApp DataOps 工具包与 JupyterHub 结合使用

这 "适用于 Kubernetes 的NetApp DataOps 工具包"可以与 JupyterHub 结合使用。通过将NetApp DataOps Toolkit 与 JupyterHub 结合使用，最终用户可以直接在 Jupyter Notebook 中创建用于工作区备份和/或数据集到模型可追溯性的卷快照。

初始设置

在将 DataOps Toolkit 与 JupyterHub 一起使用之前，您必须向 JupyterHub 分配给各个用户 Jupyter Notebook Server pod 的 Kubernetes 服务帐户授予适当的权限。 JupyterHub 使用由 `singleuser.serviceAccountName`JupyterHub Helm 图表配置文件中的变量。

为 DataOps Toolkit 创建集群角色

首先，创建一个名为“netapp-dataops”的集群角色，该角色具有创建卷快照所需的 Kubernetes API 权限。

```
$ 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
```

将集群角色分配给笔记本服务器服务帐户

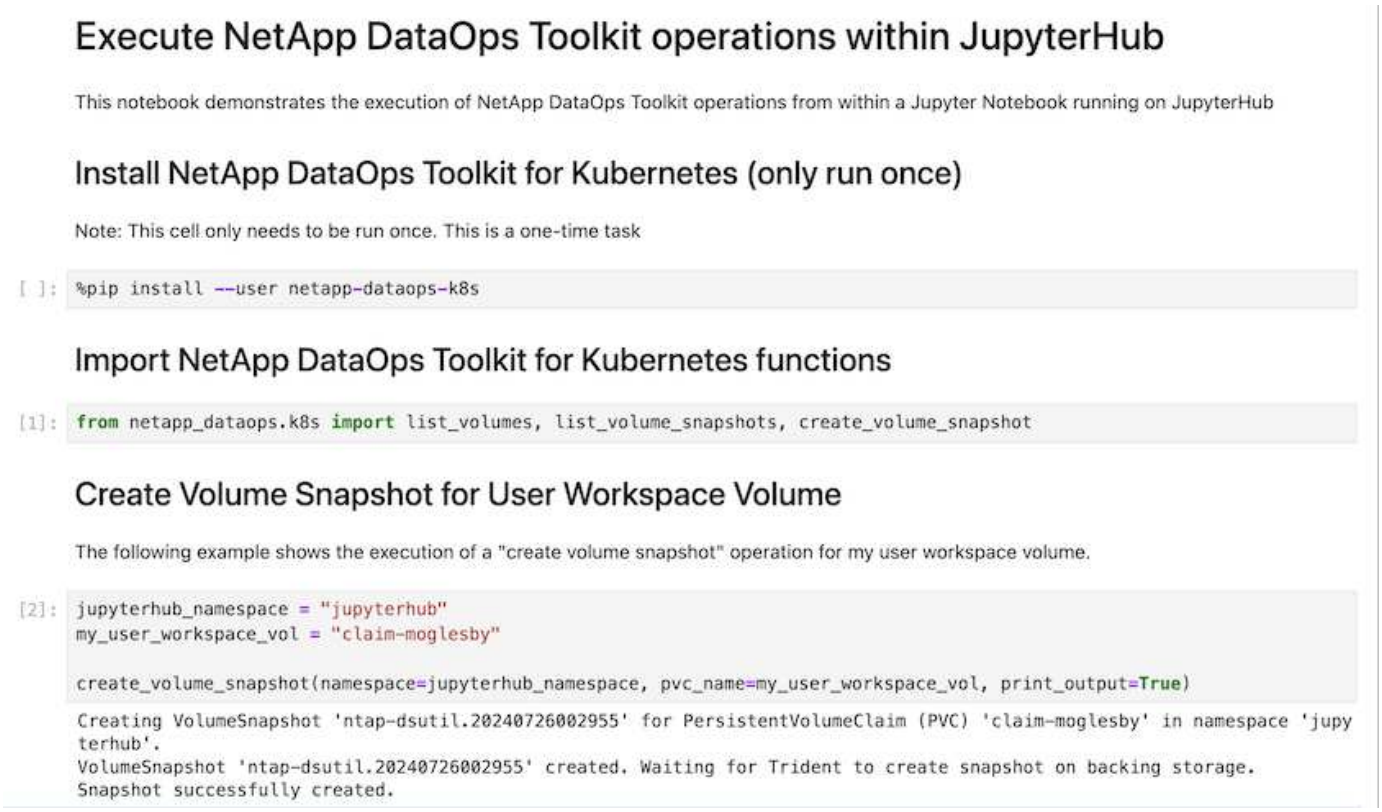
创建一个角色绑定，将“netapp-dataops-snapshots”集群角色分配给适当命名空间中的适当服务帐户。例如，如果您在“jupyterhub”命名空间中安装了 JupyterHub，并且通过以下方式指定了“默认”服务帐户 `singleuser.serviceAccountName`变量，您需要将“netapp-dataops-snapshots”集群角色分配给“jupyterhub”命名空间中的“默认”服务帐户，如下例所示。

```
$ 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** 中创建卷快照

现在，JupyterHub 用户可以使用NetApp DataOps Toolkit 直接从 Jupyter Notebook 中创建卷快照，如下例所示。



使用NetApp SnapMirror将数据导入 JupyterHub

NetApp SnapMirror是一种复制技术，可让您在NetApp存储系统之间复制数据。SnapMirror可用于将数据从远程环境提取到 JupyterHub。

示例工作流程和演示

参考"此 Tech ONTAP博客文章"有关使用NetApp SnapMirror将数据导入 JupyterHub 的详细示例工作流程和演示。

机器学习流

MLflow部署

本节介绍在 Kubernetes 集群中部署 MLflow 必须完成的任务。

> 可以在 Kubernetes 以外的平台上部署 MLflow。在 Kubernetes 以外的平台上部署 MLflow 超出了本解决方案的范围。

前提条件

在执行本节概述的部署练习之前，我们假设您已经执行了以下任务：

1. 您已经有一个可以运行的 Kubernetes 集群。
2. 您已经在 Kubernetes 集群中安装并配置了NetApp Trident 。有关Trident的更多详细信息，请参阅"Trident文档"。

安装 **Helm**

MLflow 使用 Helm（Kubernetes 的流行包管理器）进行部署。在部署 MLflow 之前，必须在 Kubernetes 控制节点上安装 Helm。要安装 Helm，请按照 "安装说明"在 Helm 官方文档中。

设置默认 **Kubernetes StorageClass**

在部署 MLflow 之前，您必须在 Kubernetes 集群中指定一个默认 StorageClass。要在集群中指定默认 StorageClass，请按照"Kubeflow部署"部分。如果您已经在集群中指定了默认 StorageClass，则可以跳过此步骤。

部署 **MLflow**

满足先决条件后，您就可以使用 Helm Chart 开始 MLflow 部署。

配置 **MLflow Helm Chart** 部署。

在使用 Helm 图表部署 MLflow 之前，我们可以使用 **config.yaml** 文件将部署配置为使用NetApp Trident存储类并更改其他参数以满足我们的需求。您可以在以下位置找到 **config.yaml** 文件的示例： https://github.com/bitnami/charts/blob/main/bitnami/mlflow/values.yaml

您可以在 config.yaml 文件中的 **global.defaultStorageClass** 参数下设置Trident storageClass（例如 storageClass：“ontap-flexvol”）。

安装 Helm Chart

可以使用以下命令将 Helm 图表与 MLflow 的自定义 **config.yaml** 文件一起安装：

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

该命令通过提供的*config.yaml*文件在自定义配置中的 Kubernetes 集群上部署 MLflow。 MLflow 部署在给定的命名空间中，并通过 kubernetes 为该版本提供一个随机发布名称。

检查部署

Helm 图表部署完成后，您可以使用以下命令检查服务是否可访问：

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

将 **jupyterhub** 替换为您在部署期间使用的命名空间。

您应该会看到以下服务：

```
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
```

我们编辑了 config.yaml 文件以使用 NodePort 服务访问端口 30002 上的 MLflow。

访问 MLflow

一旦与 MLflow 相关的所有服务都启动并运行，您就可以使用给定的 NodePort 或 LoadBalancer IP 地址访问它（例如 http://10.61.181.109:30002）

使用NetApp和 MLflow 实现数据集到模型的可追溯性

这 "适用于 Kubernetes 的NetApp DataOps 工具包"可以与 MLflow 的实验跟踪功能结合使用，以实现数据集到模型或工作区到模型的可追溯性。

要实现数据集到模型或工作区到模型的可追溯性，只需在训练运行过程中使用 DataOps Toolkit 创建数据集或工作区卷的快照，如以下示例代码片段所示。此代码将数据卷名称和快照名称保存为与您记录到 MLflow 实验跟踪服务器的特定训练运行相关的标签。

```
...
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

本节介绍在 Kubernetes 集群中部署 Kubeflow 必须完成的任务。

前提条件

在执行本节概述的部署练习之前，我们假设您已经执行了以下任务：

1. 您已经有一个可运行的 Kubernetes 集群，并且您正在运行您打算部署的 Kubeflow 版本支持的 Kubernetes 版本。有关受支持的 Kubernetes 版本列表，请参阅 Kubeflow 版本的依赖项"Kubeflow 官方文档"。
2. 您已经在 Kubernetes 集群中安装并配置了NetApp Trident 。有关Trident的更多详细信息，请参阅"Trident文档"。

设置默认 **Kubernetes StorageClass**

在部署 Kubeflow 之前，我们建议在 Kubernetes 集群中指定一个默认 StorageClass。 Kubeflow 部署过程可能会尝试使用默认 StorageClass 配置新的持久卷。如果没有指定 StorageClass 作为默认 StorageClass，则部署可能会失败。要在集群中指定默认 StorageClass，请从部署跳转主机执行以下任务。如果您已经在集群中指定了默认 StorageClass，则可以跳过此步骤。

1. 将现有 StorageClass 之一指定为默认 StorageClass。以下示例命令显示了名为 `ontap-ai-flexvols-retain`作为默认的 StorageClass。

这 `ontap-nas-flexgroup`Trident Backend 类型的最小 PVC 尺寸相当大。默认情况下，Kubeflow 尝试配置大小仅为几 GB 的 PVC。因此，您不应该指定使用 `ontap-nas-flexgroup`后端类型作为 Kubeflow 部署的默认 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
```

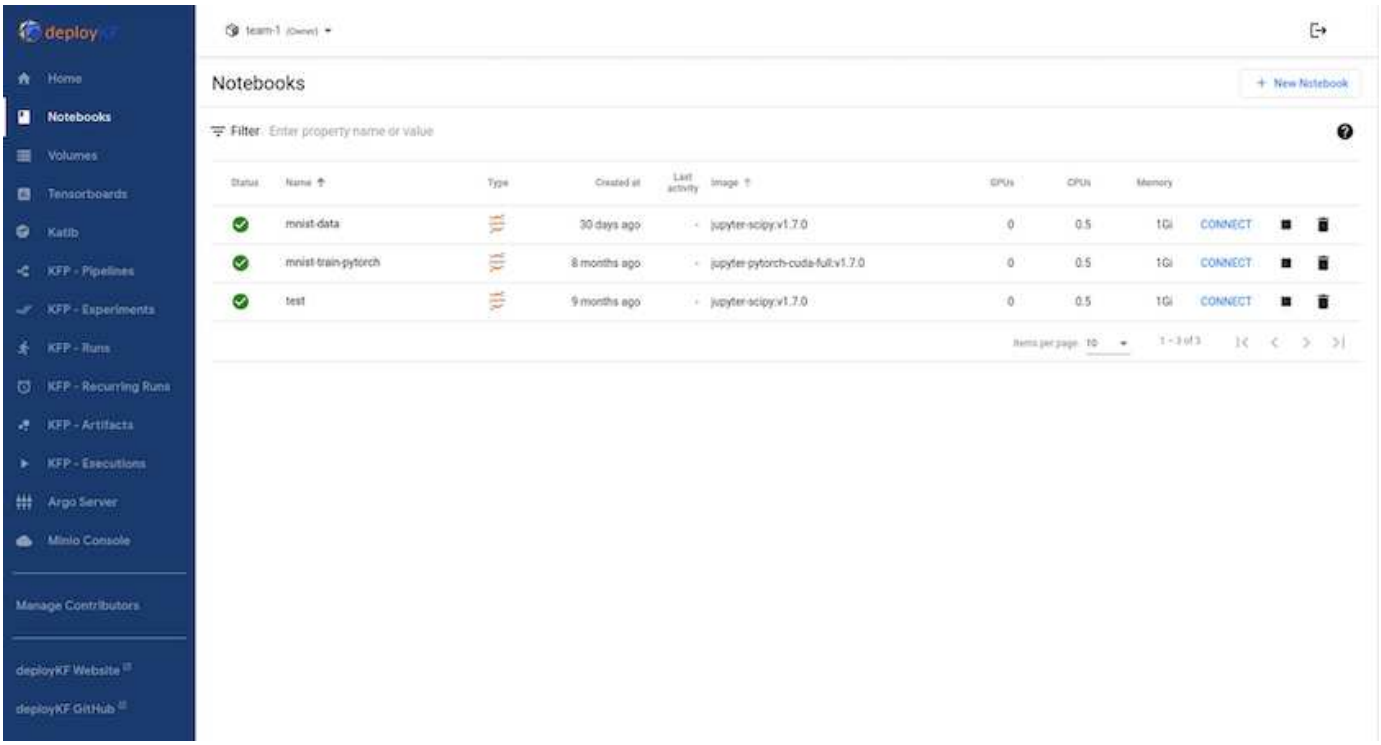
Kubeflow部署选项

部署 Kubeflow 有很多不同的选择。请参阅"Kubeflow 官方文档"获取部署选项列表，然后选择最适合您需求的选项。

为了验证目的，我们使用以下方式部署了 Kubeflow 1.7"部署KF" 0.1.1。

为数据科学家或开发人员提供 **Jupyter Notebook** 工作区

Kubeflow 能够快速配置新的 Jupyter Notebook 服务器作为数据科学家工作区。有关 Kubeflow 上下文中的 Jupyter Notebooks 的更多信息，请参阅 "Kubeflow 官方文档"。



将NetApp DataOps 工具包与 Kubeflow 结合使用

这 "适用于 Kubernetes 的NetApp数据科学工具包"可以与Kubeflow结合使用。将NetApp数据科学工具包与 Kubeflow 结合使用可带来以下好处：

- 数据科学家可以直接在 Jupyter Notebook 中执行高级NetApp数据管理操作，例如创建快照和克隆。
- 可以使用 Kubeflow Pipelines 框架将高级NetApp数据管理操作（例如创建快照和克隆）纳入自动化工作流程。

请参阅 "Kubeflow 示例"有关将该工具包与 Kubeflow 结合使用的详细信息，请参阅NetApp数据科学工具包 GitHub 存储库中的部分。

示例工作流程 - 使用 Kubeflow 和NetApp DataOps 工具包训练图像识别模型

本节介绍使用 Kubeflow 和NetApp DataOps Toolkit 训练和部署用于图像识别的神经网络的步骤。这旨在作为示例来展示结合NetApp存储的训练作业。

前提条件

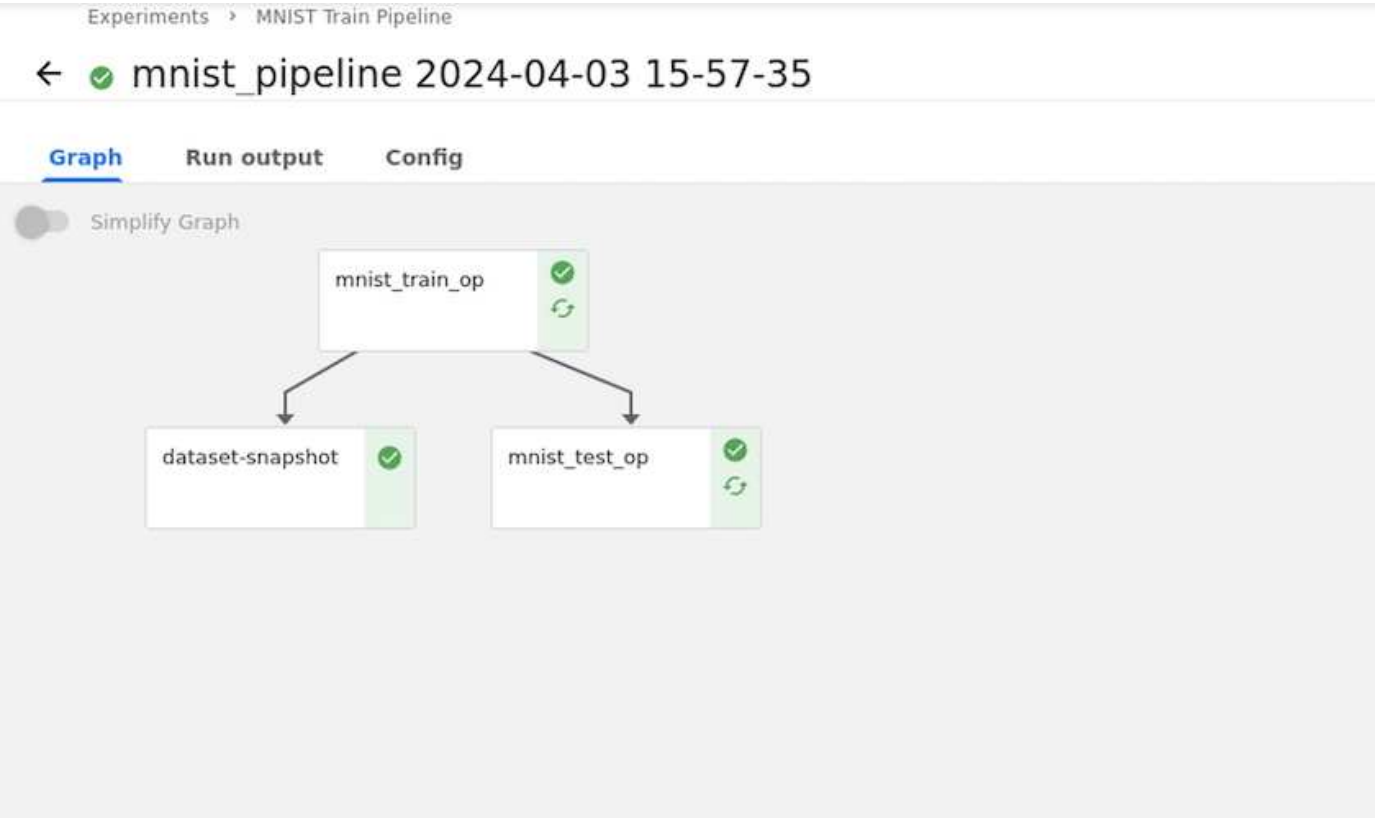
创建一个包含所需配置的 Dockerfile，用于 Kubeflow 管道内的训练和测试步骤。以下是 Dockerfile 的一个示例 -

```
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"]
```

根据您的要求，安装运行程序所需的所有必需库和包。在训练机器学习模型之前，假设您已经有一个可运行的 Kubeflow 部署。

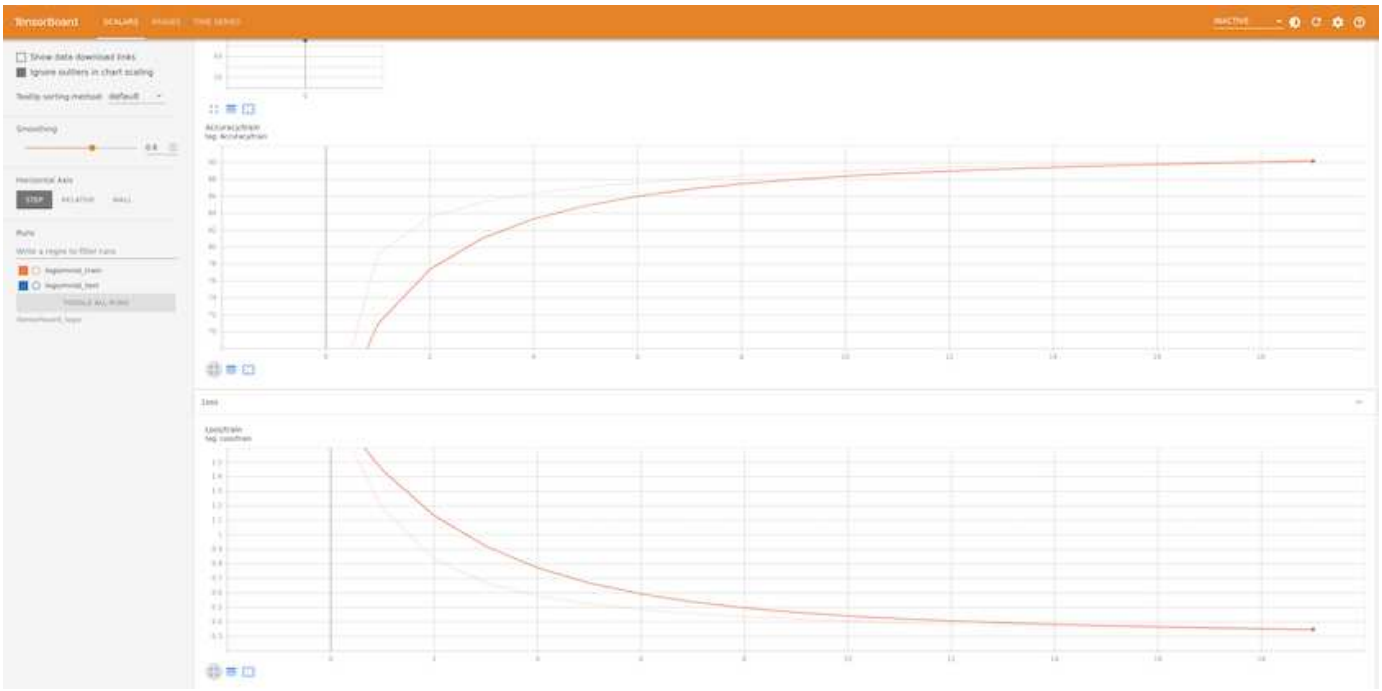
使用 **PyTorch** 和 **Kubeflow Pipelines** 在 **MNIST** 数据上训练小型 **NN**

我们使用在 MNIST 数据上训练的小型神经网络作为示例。 MNIST 数据集由 0-9 的手写数字图像组成。图像尺寸为 28x28 像素。该数据集分为 60,000 张训练图像和 10,000 张验证图像。本实验所采用的神经网络是一个2层前馈网络。训练是使用 Kubeflow Pipelines 执行的。请参阅文档 "此处"了解更多信息。我们的 Kubeflow 管道包含了先决条件部分的 docker 镜像。



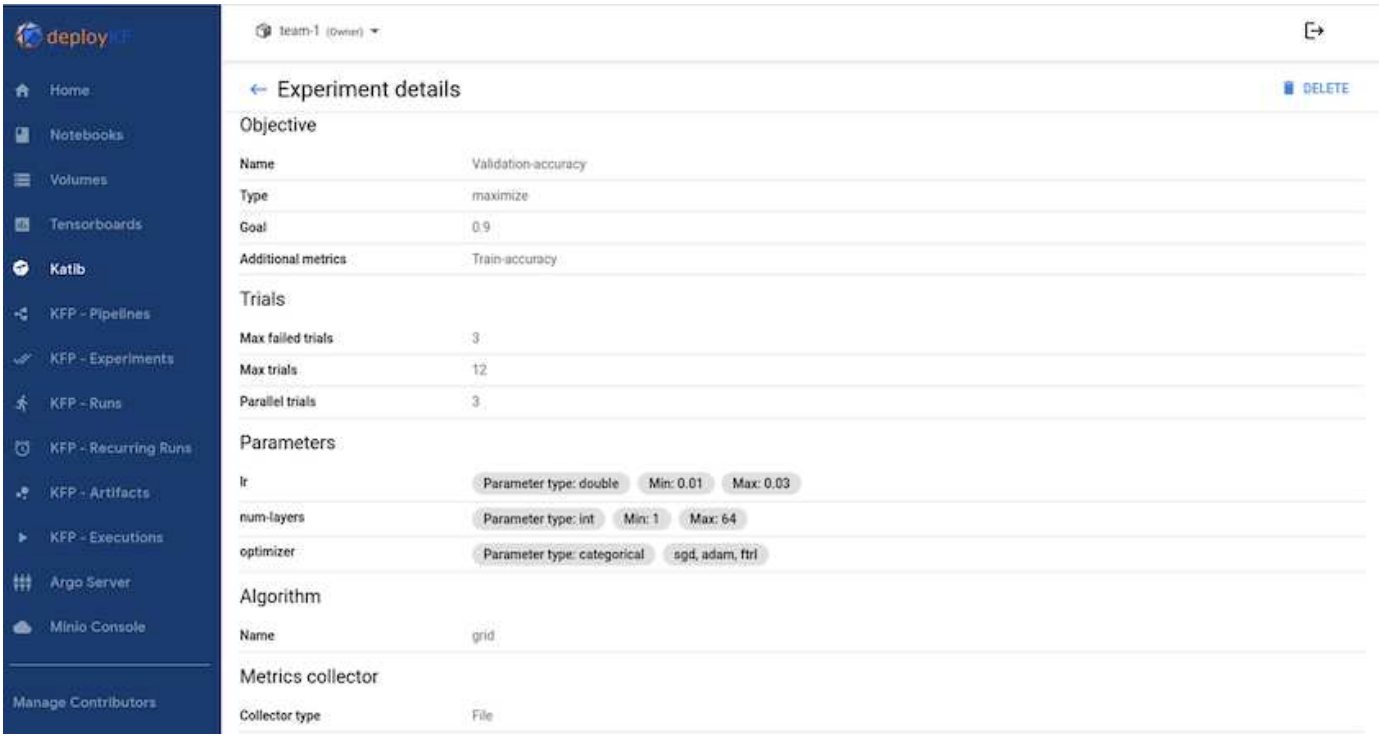
使用 **Tensorboard** 可视化结果

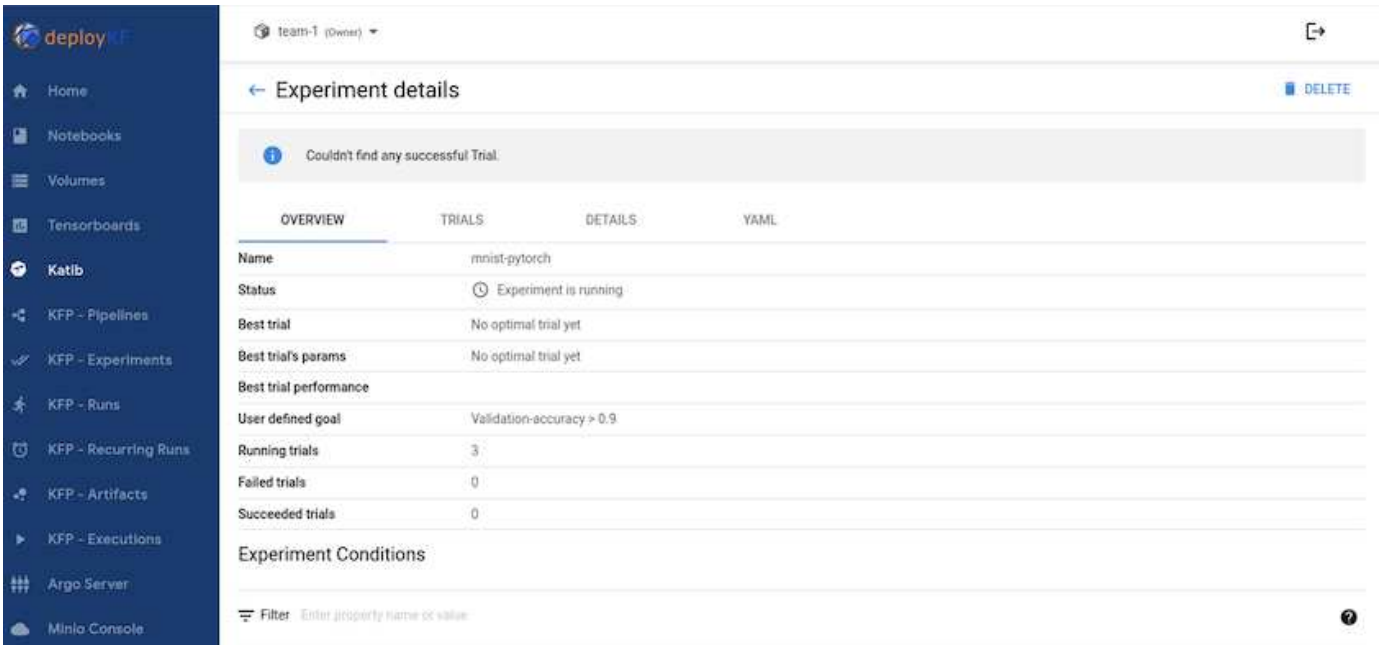
一旦模型训练完成，我们就可以使用 Tensorboard 将结果可视化。 "Tensorboard"作为 Kubeflow 仪表板上的一项功能提供。您可以为您的工作创建自定义张量板。下面的示例展示了训练准确度与时期数以及训练损失与时期数的关系图。



使用 **Katib** 进行超参数实验

"卡提布"是 Kubeflow 中的一个工具，可用于试验模型超参数。要创建实验，首先要定义所需的指标/目标。这通常是测试准确度。一旦定义了指标，选择您想要使用的超参数（优化器/学习率/层数）。 Katib 使用用户定义的值进行超参数扫描，以找到满足所需指标的最佳参数组合。您可以在 UI 的每个部分中定义这些参数。或者，您可以定义一个具有必要规范的 **YAML** 文件。以下是 Katib 实验的说明 -





使用**NetApp**快照保存数据以实现可追溯性

在模型训练期间，我们可能希望保存训练数据集的快照以便于追溯。为此，我们可以向管道添加快照步骤，如下所示。要创建快照，我们可以使用 "适用于 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)
```

请参阅 "适用于 Kubeflow 的NetApp DataOps Toolkit 示例"了解更多信息。

Trident操作示例

本节包含您可能想要使用Trident执行的各种操作的示例。

导入现有卷

如果您的NetApp存储系统/平台上存在现有卷，并且您想要将其安装在 Kubernetes 集群内的容器上，但这些卷未与集群中的 PVC 绑定，则必须导入这些卷。您可以使用Trident卷导入功能来导入这些卷。

以下示例命令显示导入名为 `pb_fg_all`。有关 PVC 的更多信息，请参阅 "Kubernetes 官方文档"。有关卷导入功能的更多信息，请参阅 "Trident文档"。

一个 `accessModes`的价值 `ReadOnlyMany`在示例 PVC 规范文件中指定。有关 `accessMode`字段，请参阅 "Kubernetes 官方文档"。

```
$ 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存储系统或平台上配置新卷。

使用 **kubectl** 配置新卷

以下示例命令显示使用 kubectl 配置新的FlexVol volume。

一个 `accessModes`的价值 `ReadWriteMany`在下面的示例 PVC 定义文件中指定。有关 `accessMode`字段，请参阅 "Kubernetes 官方文档"。

```
$ 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** 工具包配置新卷

您还可以使用NetApp DataOps Toolkit for Kubernetes 在NetApp存储系统或平台上配置新卷。 NetApp DataOps Toolkit for Kubernetes 利用Trident来配置卷，但简化了用户的流程。请参阅"文档"了解详情。

AIPod部署的高性能作业示例

执行单节点 AI 工作负载

要在 Kubernetes 集群中执行单节点 AI 和 ML 作业，请从部署跳转主机执行以下任务。使用Trident，您可以快速轻松地创建可能包含 PB 级数据的数据卷，以供 Kubernetes 工作负载访问。为了使此类数据卷可从 Kubernetes pod 内部访问，只需在 pod 定义中指定 PVC。

本节假设您已经将尝试在 Kubernetes 集群中执行的特定 AI 和 ML 工作负载容器化（以 Docker 容器格式）。

1. 以下示例命令展示了如何为使用 ImageNet 数据集的 TensorFlow 基准工作负载创建 Kubernetes 作业。有关 ImageNet 数据集的更多信息，请参阅 "ImageNet 网站"。

   此示例作业请求八个 GPU，因此可以在具有八个或更多 GPU 的单个 GPU 工作节点上运行。此示例作业可以在集群中提交，该集群中不存在具有八个或更多 GPU 的工作节点，或者当前正被另一个工作负载占用。如果是，那么该作业将保持待处理状态，直到有这样的工作节点可用。

   此外，为了最大限度地提高存储带宽，包含所需训练数据的卷在该作业创建的 pod 中被安装了两次。另一个卷也安装在 pod 中。第二卷将用于存储结果和指标。这些卷在作业定义中通过使用 PVC 的名称来引用。有关 Kubernetes 作业的更多信息，请参阅 "Kubernetes 官方文档"。

   一个 `emptyDir`音量 `medium`的价值 `Memory`安装到 `/dev/shm`在此示例作业创建的 pod 中。默认大小 `/dev/shm`Docker 容器运行时自动创建的虚拟卷有时无法满足 TensorFlow 的需求。安装 `emptyDir`如下例所示，音量提供了足够大的 `/dev/shm`虚拟卷。有关更多信息 `emptyDir`卷，参见 "Kubernetes 官方文档"。

   此示例作业定义中指定的单个容器被赋予 `securityContext > privileged`的价值 `true`。该值意味着容器实际上在主机上具有 root 访问权限。在这种情况下使用此注释，因为正在执行的特定工作负载需要 root 访问权限。具体来说，工作负载执行的清除缓存操作需要 root 访问权限。不管这是否 `privileged: true`注释是否必要取决于您正在执行的特定工作负载的要求。

```
$ 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. 确认您在步骤 1 中创建的作业正在正确运行。以下示例命令确认已为该作业创建了一个 pod（如作业定义中所指定），并且该 pod 当前正在其中一个 GPU 工作节点上运行。

```
$ 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. 确认您在步骤 1 中创建的作业已成功完成。以下示例命令确认作业已成功完成。

```
$ 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. *可选：*清理工作成果。以下示例命令显示删除在步骤 1 中创建的作业对象。

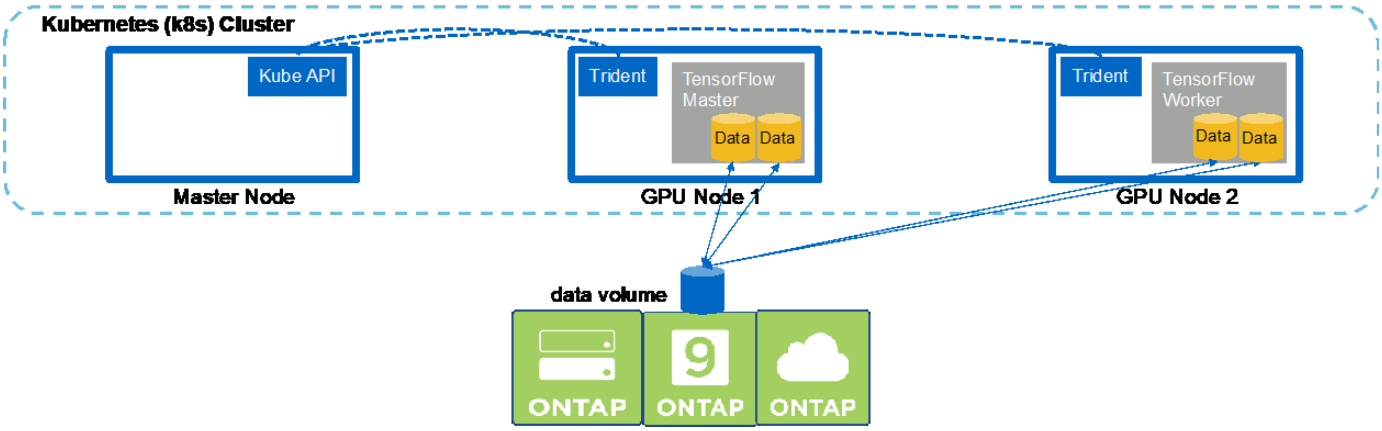
   当您删除作业对象时，Kubernetes 会自动删除任何关联的 pod。

```
$ 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.
```

执行同步分布式 AI 工作负载

要在 Kubernetes 集群中执行同步多节点 AI 和 ML 作业，请在部署跳转主机上执行以下任务。此过程使您能够利用存储在NetApp卷上的数据，并使用比单个工作节点所能提供的更多的 GPU。请参阅下图了解同步分布式 AI 作业的描述。

与异步分布式作业相比，同步分布式作业可以帮助提高性能和训练准确性。关于同步作业与异步作业的优缺点的讨论超出了本文档的范围。



1. 以下示例命令显示了如何创建一个工作器，该工作器参与本节示例中在单个节点上执行的同一 TensorFlow 基准测试作业的同步分布式执行"执行单节点 AI 工作负载"。在这个特定的例子中，只部署了一个工作器，因为作业是在两个工作器节点上执行的。

   此示例工作器部署请求八个 GPU，因此可以在具有八个或更多 GPU 的单个 GPU 工作器节点上运行。如果您的 GPU 工作节点具有超过 8 个 GPU，为了最大限度地提高性能，您可能需要将此数字增加到等于您的工作节点所具有的 GPU 数量。有关 Kubernetes 部署的更多信息，请参阅 "Kubernetes 官方文档"。

在此示例中创建了 Kubernetes 部署，因为这个特定的容器化工作程序永远无法自行完成。因此，使用 Kubernetes 作业构造来部署它是没有意义的。如果您的工作者被设计或编写为自行完成，那么使用作业构造来部署您的工作者可能是有意义的。

此示例部署规范中指定的 pod 被赋予 `hostNetwork`的价值 `true`。此值意味着 pod 使用主机工作节点的网络堆栈，而不是 Kubernetes 通常为每个 pod 创建的虚拟网络堆栈。在这种情况下使用此注释，因为特定的工作负载依赖于 Open MPI、NCCL 和 Horovod 以同步分布式方式执行工作负载。因此，它需要访问主机网络堆栈。有关 Open MPI、NCCL 和 Horovod 的讨论超出了本文档的范围。不管这是否 `hostNetwork: true`注释是否必要取决于您正在执行的特定工作负载的要求。有关 `hostNetwork`字段，请参阅 "Kubernetes 官方文档"。

```
$ 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. 确认您在步骤 1 中创建的工作程序部署已成功启动。以下示例命令确认已为部署创建了一个工作程序 pod（如部署定义中所示），并且该 pod 当前正在其中一个 GPU 工作程序节点上运行。

```
$ 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. 为主服务器创建一个 Kubernetes 作业，该主服务器启动、参与并跟踪同步多节点作业的执行。以下示例命令创建一个主服务器，该主服务器启动、参与并跟踪在本节示例中在单个节点上执行的相同 TensorFlow 基准测试作业的同步分布式执行"执行单节点 AI 工作负载"。

   此示例主作业请求八个 GPU，因此可以在具有八个或更多 GPU 的单个 GPU 工作节点上运行。如果您的 GPU 工作节点具有超过 8 个 GPU，为了最大限度地提高性能，您可能需要将此数字增加到等于您的工作节点所具有的 GPU 数量。

   此示例作业定义中指定的主 Pod 被赋予 `hostNetwork`的价值 `true`就像工作舱被赋予了 `hostNetwork`的价值 `true`在步骤 1 中。有关为什么需要此值的详细信息，请参阅步骤 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. 确认您在步骤 3 中创建的主作业正在正确运行。以下示例命令确认已为该作业创建了一个主 pod（如作业定义中所示），并且该 pod 当前正在其中一个 GPU 工作节点上运行。您还应该看到，您在步骤 1 中最初看到的工作 pod 仍在运行，并且主 pod 和工作 pod 在不同的节点上运行。

```
$ 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. 确认您在步骤 3 中创建的主作业已成功完成。以下示例命令确认作业已成功完成。

```
$ 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. 当您不再需要工作部署时，请删除它。以下示例命令显示删除在步骤 1 中创建的工作程序部署对象。

   当您删除工作部署对象时，Kubernetes 会自动删除任何关联的工作容器。

```
$ 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. *可选：*清理主作业工件。以下示例命令显示删除在步骤 3 中创建的主作业对象。

   当您删除主作业对象时，Kubernetes 会自动删除任何关联的主 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.
```