NetApp的开源MLOps

NetApp Solutions

NetApp
May 10, 2024

目录

NetApp的开源MLOps

NetApp的开源MLOps

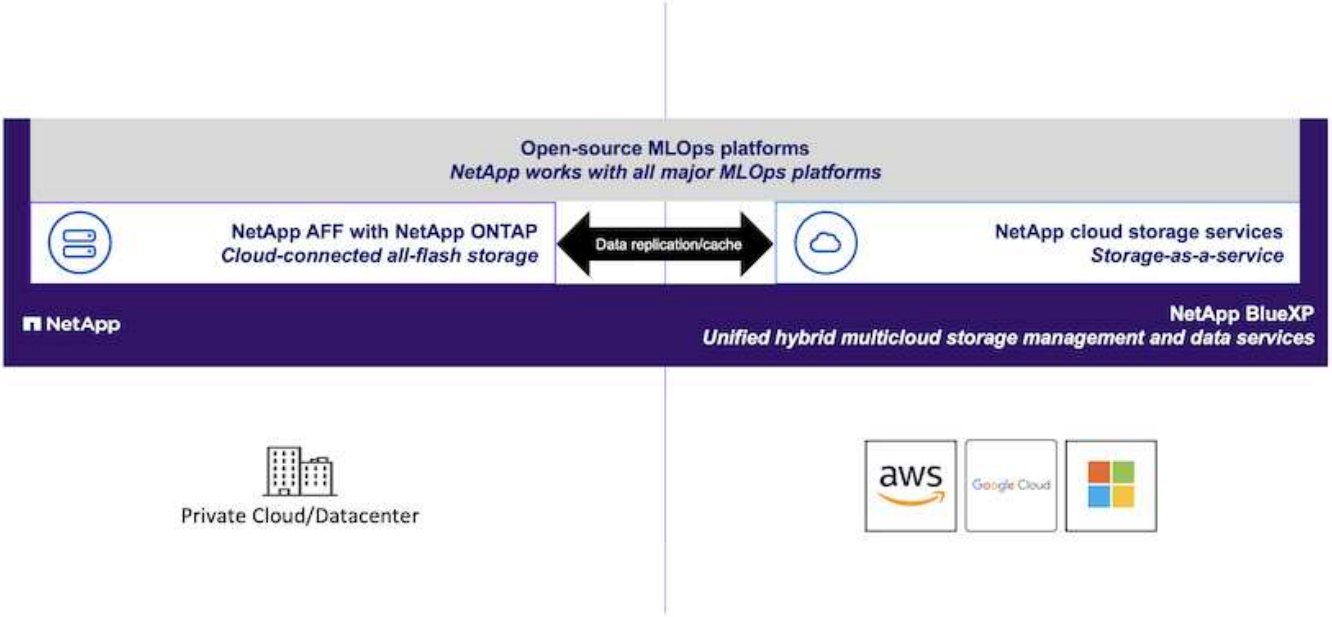
NetApp 公司 Mike Oglesby
NetApp公司的Mohan Acharya先生

各行各业各种规模的企业和组织都在转向人工智能（AI），机器学习（ML）和深度学习（DL），以解决实际问题，提供创新产品和服务，并在竞争日益激烈的市场中占据优势。随着企业越来越多地使用 AI， ML 和 DL，他们面临着许多挑战，包括工作负载可扩展性和数据可用性。本解决方案演示了如何通过将NetApp数据管理功能与常用开源工具和框架结合使用来应对这些挑战。

本解决方案旨在演示可整合到MLOps工作流中的多种不同开源工具和框架。这些不同的工具和框架可以结合使用、也可以单独使用、具体取决于要求和用例。

此解决方案涵盖以下工具/框架：

- "Apache 气流"
- "Kubeflow"



技术概述

人工智能

AI 是一门计算机科学学科，其中计算机经过训练，可以模拟人类思维的认知功能。AI 开发人员培训计算机，以便以类似于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 AI 的子领域。企业越来越多地采用 AI， ML 和 DL 来满足其关键业务需求。以下是一些示例：

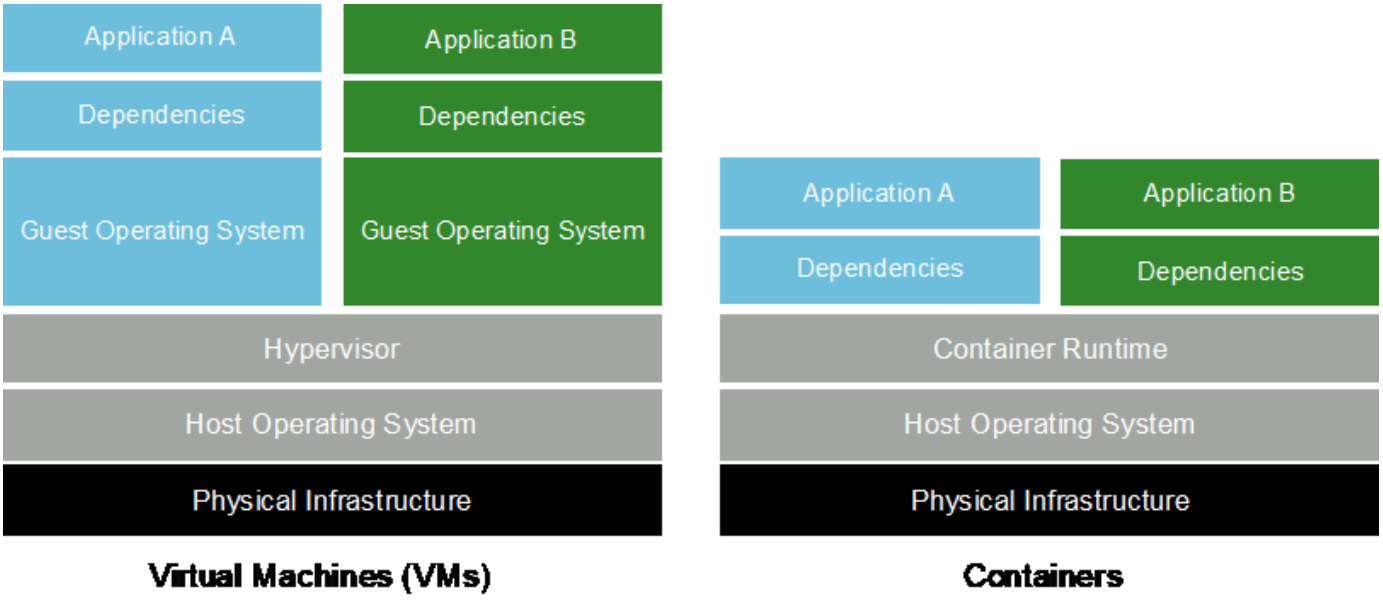
- 分析大量数据以挖掘以前未知的业务洞察力
- 使用自然语言处理直接与客户互动
- 自动化执行各种业务流程和功能

现代 AI 训练和推理工作负载需要大规模并行计算功能。因此， GPU 越来越多地用于执行 AI 操作，因为 GPU 的并行处理功能远远优于通用 CPU 。

容器

容器是在共享主机操作系统内核上运行的隔离用户空间实例。容器的采用率正在快速增长。容器可提供许多与虚拟机（ VM ）相同的应用程序沙盒优势。但是，由于虚拟机所依赖的虚拟机管理程序和子操作系统层已被消除，因此容器的重量要轻得多。下图展示了虚拟机与容器的可视化情况。

此外，还可以通过容器直接将应用程序依赖关系，运行时间等内容高效地打包到应用程序中。最常用的容器打包格式是 Docker 容器。已采用 Docker 容器格式进行容器化的应用程序可以在可以运行 Docker 容器的任何计算机上执行。即使计算机上不存在应用程序的依赖关系，也是如此，因为所有依赖关系都打包在容器中。有关详细信息，请访问 "Docker 网站"。



Kubernetes

Kubernetes 是一款开源分布式容器编排平台，最初由 Google 设计，现在由 Cloud 原生计算基金会（ CNCF ）维护。Kubernetes 可以为容器化应用程序实现部署，管理和扩展功能的自动化。近年来， Kubernetes 已成为主导容器业务流程平台。有关详细信息，请访问 "Kubernetes 网站"。

NetApp Astra Trident

Asta Trident支持在公有云或内部环境中的所有常见NetApp存储平台上使用和管理存储资源、包括ONTAP (AFF、FAS、Select、云、 Amazon FSx for NetApp ONTAP)、Element软件(NetApp HCI、SolidFire)、Azure NetApp Files服务以及Google Cloud上的Cloud Volumes Service。Asta Trident是一款符合容器存储接口(CSI)的动态存储编排程序、可与Kubbernetes本机集成。

NetApp DataOps 工具包

。 "NetApp DataOps 工具包" 是一款基于Python的工具、可简化由高性能横向扩展NetApp存储提供支持的开发/培训工作空间和参考服务器的管理。主要功能包括：

- 快速配置以高性能横向扩展NetApp存储为后盾的新的高容量工作空间。
- 近乎即时地克隆高容量工作空间、以便进行实验或快速迭代。
- 近乎即时地保存高容量工作空间的快照、以便进行备份和/或可追溯性/基线化。
- 近乎即时地配置、克隆和快照高容量、高性能数据卷。

Kubeflow

Kubeflow 是一款适用于 Kubernetes 的开源 AI 和 ML 工具包，最初由 Google 开发。通过 Kubeflow 项目，可以在 Kubernetes 上轻松，便携且可扩展地部署 AI 和 ML 工作流。Kubeflow—了Kubenetes的复杂性、使数据科学家能够专注于他们最了解的数据科学。有关可视化效果，请参见下图。对于更喜欢一体化MLOps平台的组织来说、Kubeflow是一个很好的开源选择。有关详细信息，请访问 "Kubeflow 网站"。

KubeFlow 管道

Kubeflow 管道是 Kubeflow 的一个关键组件。Kubeflow 管道是一个平台和标准，用于定义和部署可移植且可扩展的 AI 和 ML 工作流。有关详细信息，请参见 "Kubeflow 官方文档"。

Jupyter 笔记本电脑服务器

Jupyter 笔记本电脑服务器是一款开源 Web 应用程序，数据科学家可以利用它创建类似于维基的文档，这些文档称为 Jupyter 笔记本电脑，其中包含实时代码以及描述性测试。Jupyter 笔记本电脑在 AI 和 ML 社区中广泛使用，可用于记录，存储和共享 AI 和 ML 项目。Kubeflow 可简化 Kubernetes 上 Jupyter 笔记本电脑服务器的配置和部署。有关 Jupyter 笔记本电脑的详细信息，请访问 "Jupyter 网站"。有关 Kubeflow 环境中 Jupyter 笔记本电脑的详细信息，请参见 "Kubeflow 官方文档"。

Katb.

Katis是一个用于自动化机器学习(AutoML)的Kubornetes原生项目。Katb支持超参数调整、早期停止和神经架构搜索(NAS)。Katis是一个与机器学习(ML)框架无关的项目。它可以调整以用户选择的任何语言编写的应用程序的超参数、并本机支持许多ML框架、例如TensorFlow、MXNet、PyTorch、XGBoost、 等。Katib"支持许多各种AutoML算法、例如贝叶斯优化、帕森树估算器、随机搜索、协方差矩阵自适应进化策略、超频带、高效神经架构搜索、可区分架构搜索等。有关 Kubeflow 环境中 Jupyter 笔记本电脑的详细信息，请参见 "Kubeflow 官方文档"。

Apache 气流

Apache Airflow 是一个开源工作流管理平台，支持对复杂的企业工作流进行编程创作，计划和监控。它通常用于自动执行 ETL 和数据管道工作流，但不限于这些类型的工作流。气流项目由 Airbnb 发起，但此后在业内备受欢迎，现在由 Apache 软件基金会赞助。气流采用 Python 编写，气流工作流通过 Python 脚本创建，气流是按照 " 配置即代码 " 原则设计的。 现在，许多企业级气流用户都在 Kubernetes 上运行 Airflow 。

定向循环图（ DAG ）

在气流模式下，工作流称为定向环比图（ DAG ）。DAG 由按顺序，并行或两者的组合执行的任务组成，具体取决于 DAG 定义。气流计划程序会根据 DAG 定义中指定的任务级别依赖关系对一组工作负载执行各个任务。DAG 是通过 Python 脚本定义和创建的。

NetApp ONTAP

ONTAP 9是NetApp推出的最新一代存储管理软件、可帮助企业打造现代化的基础架构并过渡到云就绪数据中心。借助行业领先的数据管理功能，无论数据位于何处， ONTAP 都可以通过一组工具来管理和保护数据。您还可以将数据自由移动到需要的任何位置：边缘，核心或云。ONTAP 9包含许多功能、可简化数据管理、加快和保护关键数据、并在混合云架构中实现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、这样才能将适当的资源用于AI应用程序和训练AI/ML数据集。以下有关NetApp技术的追加信息 不在此验证范围内、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：

- 实时数据缩减和扩展的重复数据删除。数据缩减可减少存储块中浪费的空间、重复数据删除可显著提高有效容量。此适用场景数据存储在本地，并分层到云。
- 最低、最高和自适应服务质量(AQoS)。精细的服务质量(QoS)控制有助于在高度共享的环境中保持关键应用程序的性能水平。
- NetApp FabricPool。可将冷数据自动分层到公有 和私有云存储选项、包括Amazon Web Services (AWS)、Azure和NetApp StorageGRID Storage解决方案。有关 FabricPool 的详细信息，请参见 "TR-4598：FabricPool 最佳实践"。

加速和保护数据

ONTAP 可提供卓越的性能和数据保护、并通过以下方式扩展这些功能：

- 性能和更低的延迟。ONTAP 可提供尽可能高的吞吐量和尽可能低的延迟。
- 数据保护ONTAP 可提供内置数据保护功能、并在所有平台之间进行通用管理。
- NetApp卷加密(NVE)。ONTAP 提供原生 卷级加密、并支持板载和外部密钥管理。
- 多租户和多因素身份验证。ONTAP 支持以最高的安全性级别共享基础架构资源。

Future-Proof 基础架构

ONTAP 可通过以下功能满足不断变化的苛刻业务需求：

- 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 NVMe 和 32 Gb FC 等最新技术，而无需进行成本高昂的数据迁移或中断。
- 云连接。ONTAP是云互联程度最高的存储管理软件、可在所有公有云中选择软件定义的存储和云原生实例。
- 与新兴应用程序集成。ONTAP 通过使用支持现有企业应用程序的相同基础架构、为下一代平台和应用程序(例如自动驾驶汽车、智能城市和行业4.0)提供企业级数据服务。

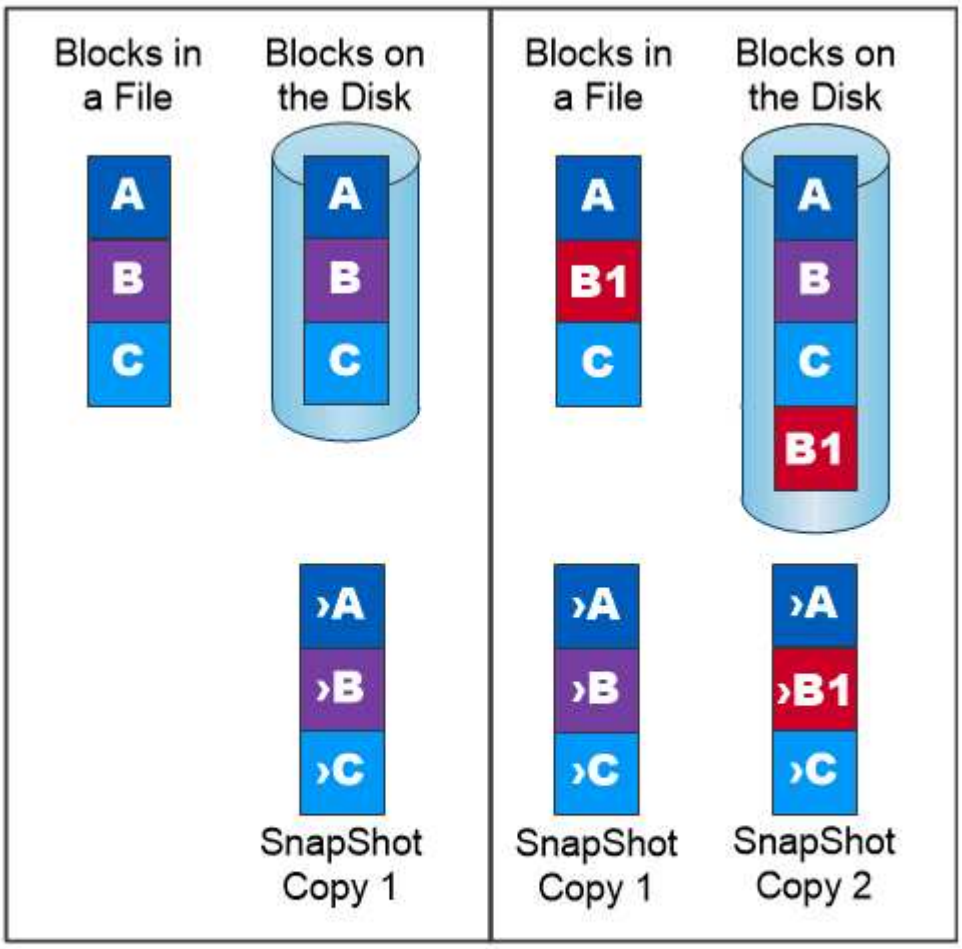
NetApp Snapshot 副本

NetApp Snapshot 副本是卷的只读时间点映像。该映像占用的存储空间极少，并且性能开销极低，因为它仅记录自创建上次 Snapshot 副本以来创建的文件所做的更改，如下图所示。

Snapshot 副本的效率归功于核心 ONTAP 存储虚拟化技术—任意位置写入文件布局（ Write Anywhere File Layout ， 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。它会将更新后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，因此 Snapshot 副本的效率非常高。这样做可以避免其他系统在查找要复制的块时花费寻道时间，并避免创建副本本身的成本。

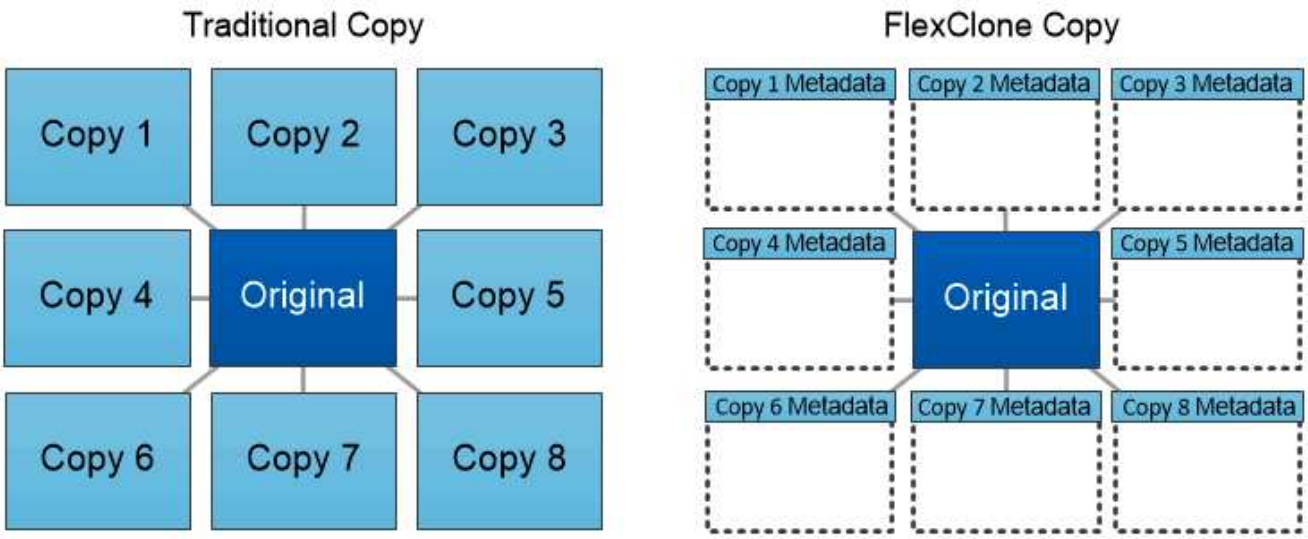
您可以使用 Snapshot 副本恢复单个文件或 LUN ，或者还原卷的整个内容。ONTAP 会将 Snapshot 副本中的指针信息与磁盘上的数据进行比较，以重建缺少或损坏的对象，而不会造成停机或高昂的性能成本。



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

NetApp FlexClone 技术

NetApp FlexClone 技术会引用 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，在将更改写入副本之前，除了元数据所需的存储外，不会占用任何其他存储，如下图所示。传统副本可能需要几分钟甚至几小时才能创建，而 FlexClone 软件可以让您几乎即时复制最大的数据集。因此，如果您需要相同数据集的多个副本（例如，开发工作空间）或数据集的临时副本（针对生产数据集测试应用程序），则这种情况是理想之选。



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 软件是一款经济高效且易于使用的统一复制解决方案，可跨数据网络结构实现。它可以通过 LAN 或 WAN 高速复制数据。它可以为各种类型的应用程序提供高数据可用性和快速数据复制，包括虚拟和传统环境中的业务关键型应用程序。在将数据复制到一个或多个 NetApp 存储系统并持续更新二级数据时，您的数据将保持最新，并可随时使用。不需要外部复制服务器。有关利用 SnapMirror 技术的架构示例，请参见下图。

SnapMirror 软件通过仅通过网络发送更改的块来利用 NetApp ONTAP 的存储效率。SnapMirror 软件还可使用内置网络压缩来加快数据传输速度，并将网络带宽利用率降低多达 70% 。借助 SnapMirror 技术，您可以利用一个精简复制数据流创建一个存储库，同时维护活动镜像和先前的时间点副本，从而将网络流量减少多达 50% 。

NetApp BlueXP复制和同步

BlueXP复制和同步是一项NetApp服务、用于快速安全地同步数据。无论您是需要在内部NFS还是SMB文件共享、NetApp StorageGRID、NetApp ONTAP S3、NetApp Cloud Volumes Service、Azure NetApp Files、AWS S3、AWS EFS、Azure Blb、 Google Cloud Storage或IBM Cloud Object Storage、BlueXP Copy and Sync可将文件快速安全地移动到您需要的位置。

数据传输完成后，即可在源和目标上完全使用。BlueXP复制和同步功能可以在触发更新时按需同步数据、也可以根据预定义的计划持续同步数据。不管怎样、BlueXP复制和同步功能只会移动增量、因此可以最大限度地减少数据复制所需的时间和资金。

BlueXP Copy and Sync是一款软件即服务(SaaS)工具、设置和使用极其简单。由BlueXP复制和同步触发的数据传输由数据代理执行。BlueXP复制和同步数据代理可以部署在AWS、Azure、Google Cloud Platform或内部环境中。

NetApp XCP

NetApp XCP 是一款基于客户端的软件，可用于任意到 NetApp 以及 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系统资源来处理大容量数据集和高性能迁移来实现扩展和最大性能。XCP 可通过生成报告的选项帮助您全面了解文件系统。

NetApp XCP 可通过一个软件包提供，该软件包支持 NFS 和 SMB 协议。XCP 包括一个适用于 NFS 数据集的 Linux 二进制文件和一个适用于 SMB 数据集的 Windows 可执行文件。

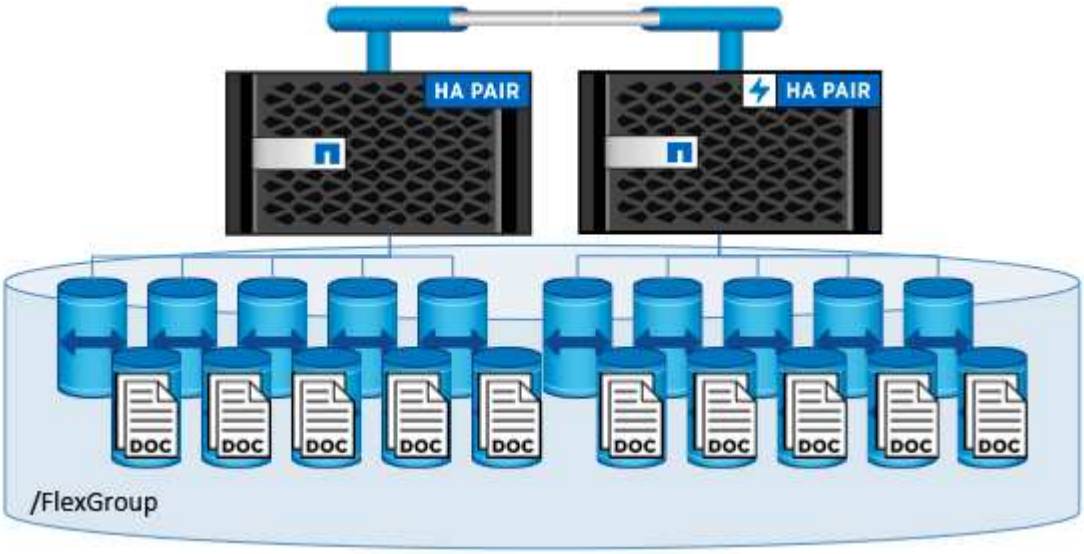
NetApp XCP 文件分析是一款基于主机的软件，可检测文件共享，对文件系统运行扫描并提供用于文件分析的信息板。XCP 文件分析与 NetApp 和非 NetApp 系统兼容，并可在 Linux 或 Windows 主机上运行，以便为 NFS 和 SMB 导出的文件系统提供分析。

NetApp ONTAP FlexGroup 卷

培训数据集可以是一组可能包含数十亿个文件的集合。文件可以包括文本，音频，视频以及其他形式的非结构化数据，这些数据必须进行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，以便执行顺序和随机 I/O

FlexGroup 卷是一个包含多个成分卷的命名空间，如下图所示。从存储管理员的角度来看， FlexGroup 卷是一个受管卷，其作用类似于 NetApp FlexVol 卷。FlexGroup 卷中的文件将分配给各个成员卷，并且不会在卷或节点之间进行条带化。它们支持以下功能：

- FlexGroup 卷可为高元数据工作负载提供多 PB 的容量和可预测的低延迟。
- 它们在同一命名空间中最多支持 4000 亿个文件。
- 它们支持在 CPU ，节点，聚合和成分卷之间的 NAS 工作负载中执行并行操作 FlexVol 。



架构

此解决方案不依赖于特定硬件。解决方案与 Trident 支持的任何 NetApp 物理存储设备，软件定义的实例或云服务兼容。例如、NetApp AFF存储系统、Amazon FSx for NetApp ONTAP、Azure NetApp Files或NetApp Cloud Volumes ONTAP实例。此外、只要Kubeflow和NetApp Asta三端技术支持所使用的Kubenetes版本、解决方案就可以在任何Kubenetes集群上实施。有关 Kubeflow 支持的 Kubernetes 版本列表，请参见 "Kubeflow 官方文档"。有关 Trident 支持的 Kubernetes 版本列表，请参见 "Trident 文档"。有关用于验证解决方案的环境的详细信息，请参见下表。

软件组件	**version**
Apache 气流	2.0.1
Apache 气流 Helm 图表	8.0.8
Kubeflow	1.7、通过部署 "部署KF" 0.1.1.
Kubernetes	1.26.
NetApp Astra Trident	23.07

支持

NetApp不为Apache Airflow、Kubeflow或Kubenetes提供企业级支持。如果您对完全受支持的MLOps平台感兴趣、 "请联系 NetApp" 关于NetApp与合作伙伴联合提供的完全受支持的MLOps解决方案。

NetApp Astra三端配置

适用于**NetApp AIPod**部署的**Asta**三叉式后端示例

在使用Asta Trident在Kubornetes集群中动态配置存储资源之前、您必须先创建一个或多个Trident后端。以下示例表示在上将此解决方案的组件部署到上时可能要创建的不同类型的后端 "NetApp AIPod"。有关后端的详细信息，请参见 "Astra Trident 文档"。

1. NetApp建议为AIPod创建启用了FlexGroup的三端。

   以下命令示例显示了如何为AIPod Storage Virtual Machine (SVM)创建启用了FlexGroup的三端。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 卷具有大小限制（截至本文撰写时，最大大小取决于特定部署）。另一方面， FlexGroup 卷可以线性扩展到高达 20 PB 和 4000 亿个文件，从而提供一个可显著简化数据管理的命名空间。因此， FlexGroup 卷最适合依赖大量数据的 AI 和 ML 工作负载。

   如果您使用的是少量数据，并且希望使用 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` 存储驱动程序的 Trident 后端。

```
$ 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还建议创建启用了的FlexVol后端。您可能希望使用FlexVol卷托管永久性应用程序、存储结果、输出、调试信息等。如果要使用 FlexVol 卷，必须创建一个或多个启用了 FlexVol 的 Trident 后端。以下命令示例显示了创建一个启用了FlexVol的三端后端。

```
$ 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 |
+-------------------------+--------------------
+---------------------------------------+--------+---------+
```

适用于NetApp AIPod部署的Kubernetes StorageClasses示例

在使用Asta三端到功能在Kubornetes集群中动态配置存储资源之前、您必须先创建一个或多个Kubornetes StorageCluster。以下示例表示在上将此解决方案的组件部署到上时可能要创建的不同类型的StorageCluster "NetApp AIPod"。有关 StorageClasses 的详细信息，请参见 "Astra Trident 文档"。

1. NetApp建议为您在一节中创建的启用了FlexGroup的三端后端创建存储类 "适用于NetApp AIPod部署的Asta三叉式后端示例"，步骤 1 。下面的示例命令显示了创建多个StorageClasses的过程、这些StorageClasses对应于本节中创建的两个示例后端 "适用于NetApp AIPod部署的Asta三叉式后端示例"，第1步-一个利用的步骤 "基于 RDMA 的 NFS" 而不是。

   为了在删除相应的 PersistentVolumeClaim （ PVC ）时不删除永久性卷，以下示例使用了 `reClaimPolicy` 值 `Retain` 。有关 `re"claimPolicy` " 字段的详细信息，请参见相关官员 "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、必须相应地在ONTAP系统上配置最大传输大小。请参见 "ONTAP 文档" 了解详细信息。

   注意：要使用基于RDMA的NFS、必须在ONTAP系统上配置基于RDMA的NFS。有关详细信息、请参见https://docs.netapp.com/us-en/ontap/nfs-rdma/[ONTAP的第2部分。

   注意：在以下示例中、未在StorageClass定义文件的StoragePoool字段中指定特定后端。

```
$ 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部分中创建的启用了 FlexVol 的 Trident 后端对应的 StorageClass "适用于AIPod部署的Asta Trendent后端示例"，第2步。下面的示例命令显示了为 FlexVol 卷创建一个 StorageClass 的过程。

   注意：在以下示例中、未在StorageClass定义文件的StoragePoool字段中指定特定后端。当您使用Kubnetes管理使用此存储类的卷时、啮合动物会尝试使用任何使用的可用后端 `ontap-nas` 驱动程序。

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

Kubeflow

Kubeflow 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

在执行本节所述的部署练习之前，我们假定您已执行以下任务：

1. 您已有一个正在运行的Kubenetes集群、并且正在运行的Kubeflow版本支持此Kubenetes版本。有关支持的Kubernetes版本的列表、请参阅中您的Kubeflow版本的依赖关系 "Kubeflow 官方文档"。
2. 您已在Kubbernetes集群中安装并配置NetApp Asta三端存储。有关Asta三项功能的更多详细信息、请参见 "Astra Trident 文档"。

设置默认 **Kubernetes StorageClass**

在部署Kubeflow之前、我们建议在Kubenetes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配置新的永久性卷。如果未将任何StorageClass指定为默认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。如果已在集群中指定默认 StorageClass ，则可以跳过此步骤。

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

`ontap-nas-flexgroup` Trident 后端类型的最小 PVC 大小相当大。默认情况下， KubeFlow 会尝试配置大小只有少数几 GB 的 PVC 。因此，在部署 Kubeflow 时，不应将利用 `ontap-nas-flexgroup` 后端类型的 StorageClass 指定为默认 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

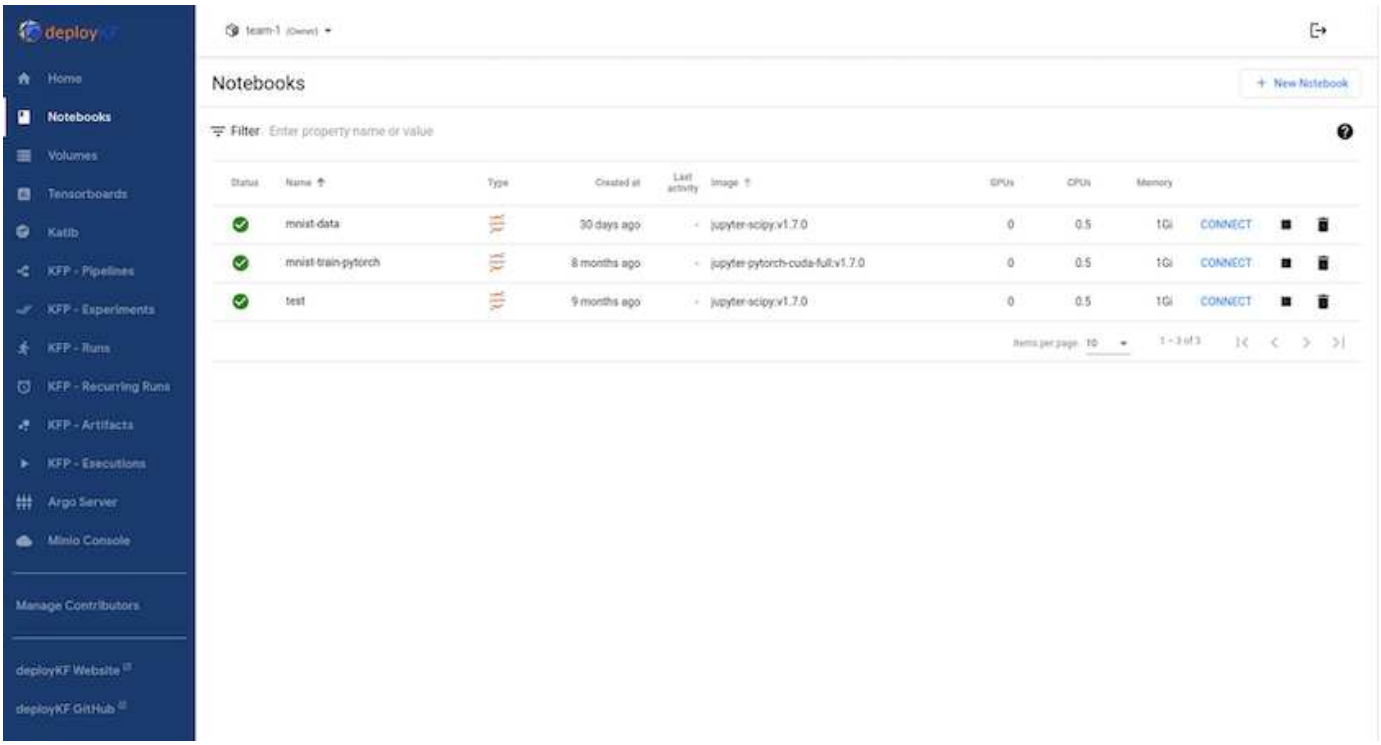
部署Kubeflow有许多不同的选项。请参见 "Kubeflow 官方文档" 有关部署选项的列表、请选择最适合您的需求的选项。

出于验证目的、我们使用部署了Kubeflow 1.7 "部署KF" 0.1.1.

Kubeflow 操作和任务示例

为数据科学家或开发人员配置 **Jupyter** 笔记本电脑工作空间

Kubeflow 能够快速配置新的 Jupyter 笔记本电脑服务器，以充当数据科学家工作空间。有关 Kubeflow 上下文中 Jupyter 笔记本电脑的详细信息，请参见 "Kubeflow 官方文档"。



将**NetApp**数据操作工具包与**Kubeflow**结合使用

。 "适用于 Kubernetes 的 NetApp 数据科学工具包" 可与 Kubeflow 结合使用。将 NetApp 数据科学工具包与 Kubeflow 结合使用具有以下优势：

- 数据科学家可以直接从Jupyter笔记本中执行高级NetApp数据管理操作、例如创建快照和克隆。
- 使用Kubeflow管道框架、可以将高级NetApp数据管理操作(例如创建快照和克隆)整合到自动化工作流中。

请参见 "Kubeflow 示例" 有关将工具包与 Kubeflow 结合使用的详细信息，请参见 NetApp Data Science Toolkit GitHub 存储库中的一节。

示例工作流-使用**Kubeflow**和**NetApp**数据操作工具包训练图像识别模型

本节介绍使用Kubeflow和NetApp数据操作工具包培训和部署用于图像识别的神经网络所涉及的步骤。此示例用于展示整合了NetApp存储的培训作业。

前提条件

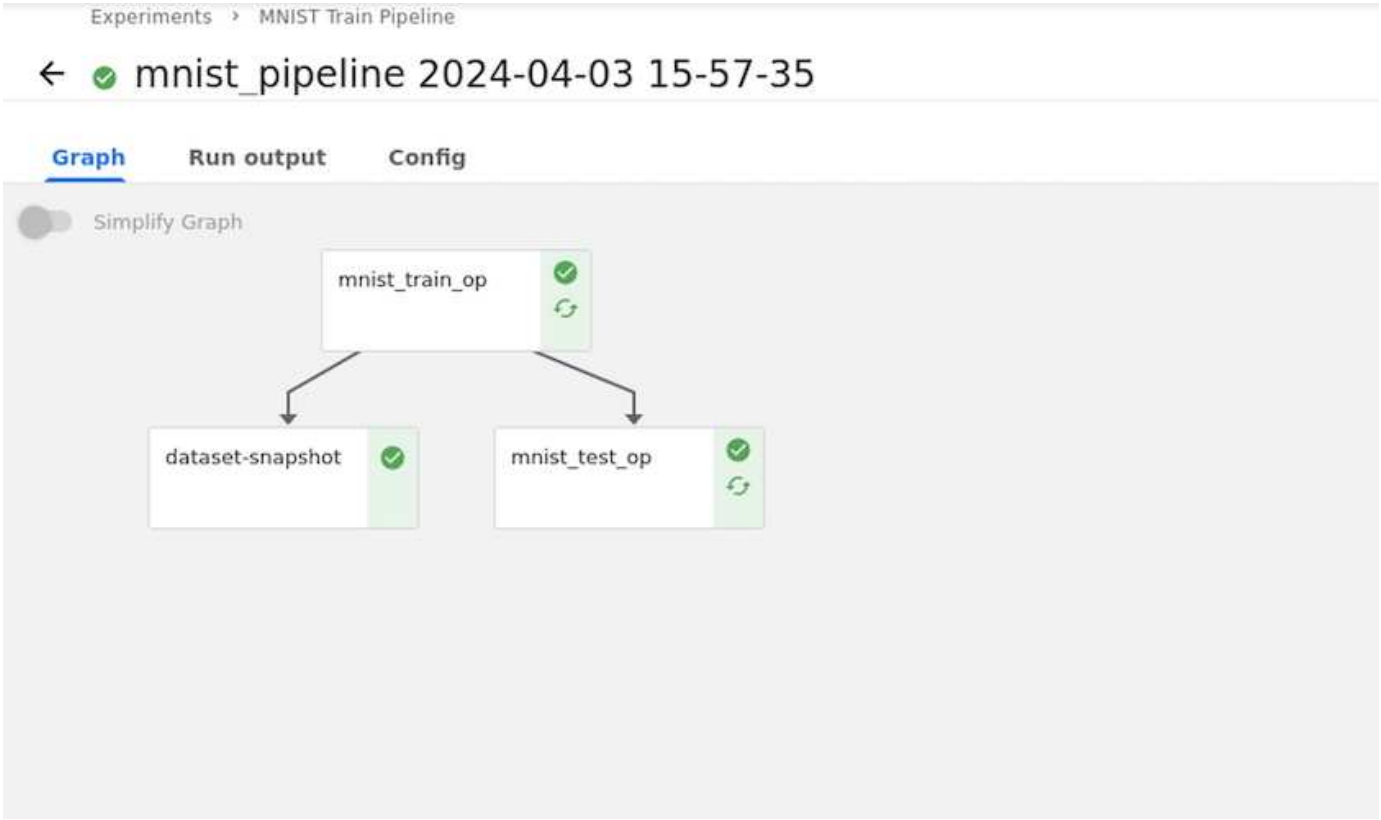
创建一个包含所需配置的文档、以用于Kubeflow管道中的训练和测试步骤。
以下是一个多克尔文件示例-

```
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，安装运行该程序所需的所有必需库和软件包。在训练机器学习模型之前、我们假定您已部署有效的Kubeflow。

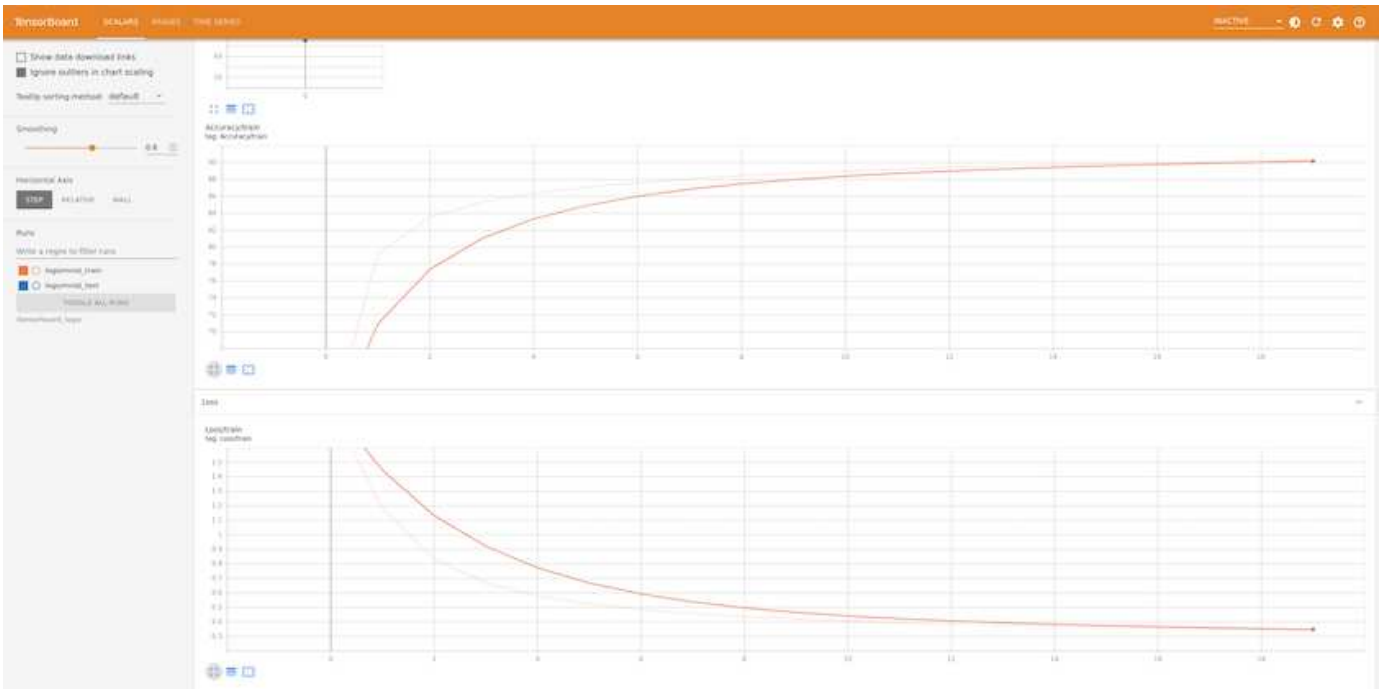
使用PyTorch和Kubeflow管道训练有关MNIST数据的小型NN

我们以一个小型神经网络为例、该网络是根据MNIST数据进行训练的。MNIST数据集由0到9位数字的字母图像组成。图像的大小为28x28像素。数据集分为60、000个训练影像和10、000个验证影像。用于此实验的神经网络是一个2层馈送网络。使用Kubeflow管道执行培训。请参见文档 "此处" 有关详细信息 …我们的Kubeflow管道整合了"前提条件"部分中的Docker映像。



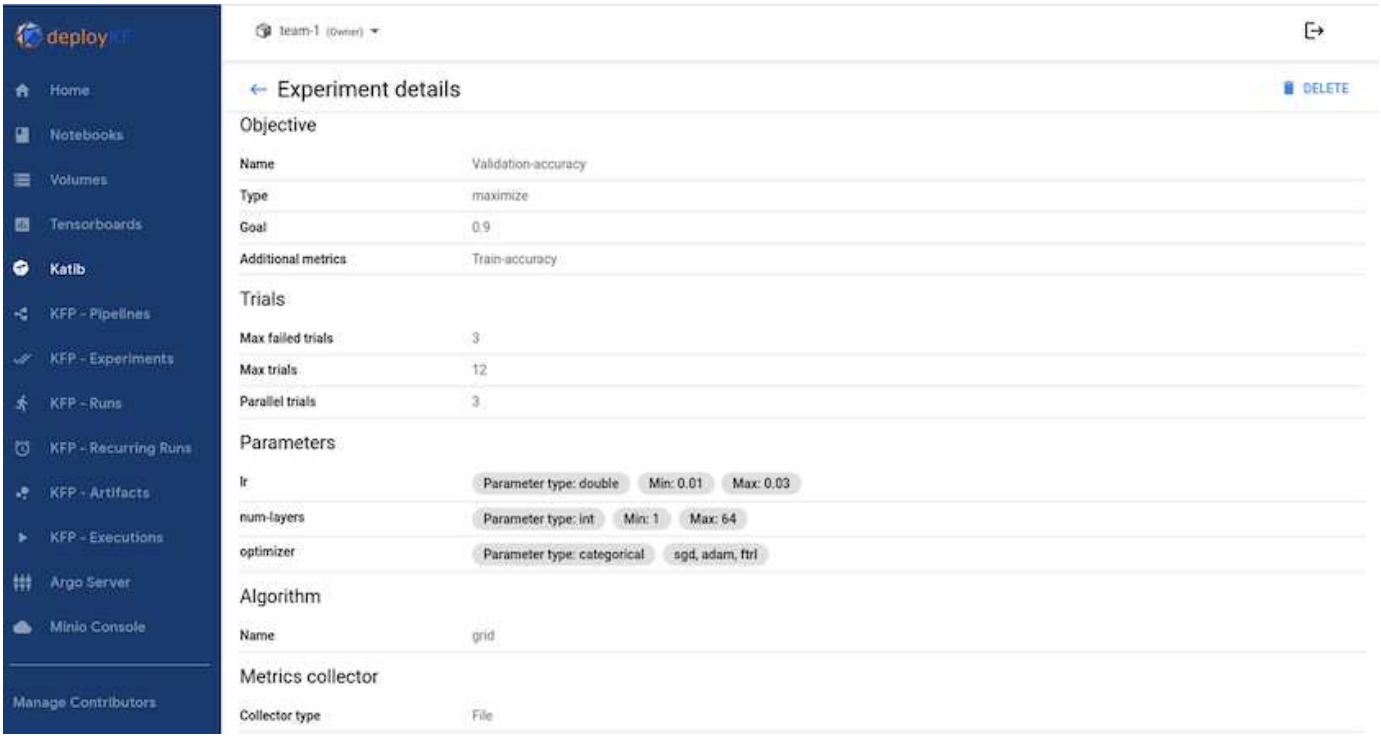
使用Tensorboard直观显示结果

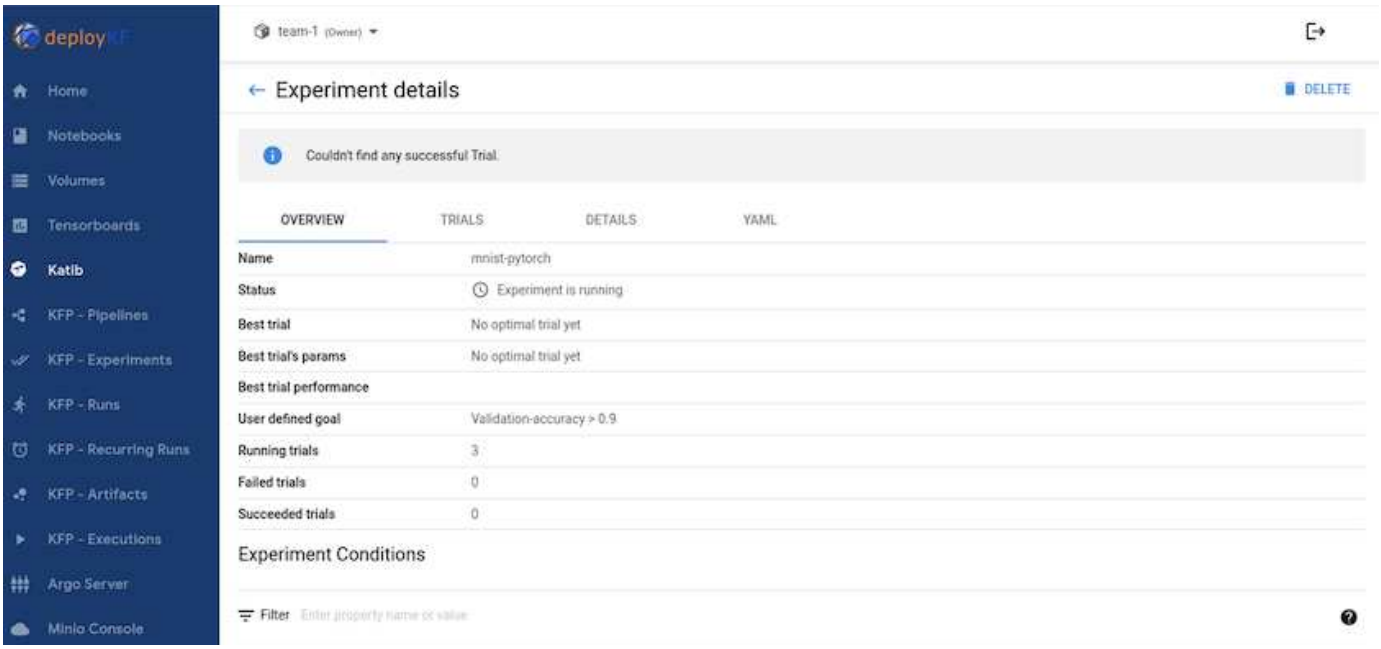
模型训练完成后、我们可以使用Tensorboard直观显示结果。 "Tensorboard" 作为Kubeflow信息板上的一项功能提供。您可以为您的作业创建自定义的tensorboard。以下示例显示了训练精度与的图解环比和训练损失的数量与时代的数量。



使用**Katb**试验超参数

"Katb." 是Kubeflow中的一个工具、可用于实验模型超参数。要创建实验、请先定义所需的指标/目标。这通常是测试准确性。定义指标后、选择要使用的超参数(优化器/leining_rate /层数)。Katb使用用户定义的值执行超参数扫描、以找到满足所需度量的最佳参数组合。您可以在用户界面的每个部分中定义这些参数。或者，也可以使用必要的规范定义*YAML*文件。下面是一个Katis实验的示意图-





使用**NetApp**快照保存数据以实现可跟踪性

在模型训练期间、我们可能希望保存训练数据集的快照、以便于跟踪。为此、我们可以向管道中添加Snapshot步骤、如下所示。要创建快照、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)
```

请参见 "适用于Kubeflow的NetApp数据操作工具包示例" 有关详细信息 …

Apache 气流

Apache Airflow 部署

本节介绍在 Kubernetes 集群中部署气流时必须完成的任务。

> 可以在 Kubernetes 以外的平台上部署 Airflow 。在 Kubernetes 以外的平台上部署气流不在此解决方案的范围内。

前提条件

在执行本节所述的部署练习之前，我们假定您已执行以下任务：

1. 您已有一个工作正常的 Kubernetes 集群。
2. 您已在Kubbernetes集群中安装并配置NetApp Asta三端存储。有关Asta三项功能的更多详细信息、请参见 "Astra Trident 文档"。

安装 **Helm**

Airflow 可使用 Kubernetes 常用的软件包管理器 Helm 进行部署。在部署气流之前，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" 在官方 Helm 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 。气流部署过程会尝试使用默认 StorageClass 配置新的永久性卷。如果未将任何 StorageClass 指定为默认 StorageClass ，则部署将失败。要在集群中指定默认StorageClass、请按照中所述的说明进行操作 "Kubeflow 部署" 部分。如果已在集群中指定默认 StorageClass ，则可以跳过此步骤。

使用 **Helm** 部署气流

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

1. 按照说明使用 Helm 部署气流 "部署说明" 用于 Artifact Hub 上的官方气流图表。下面的示例命令显示了如何使用 Helm 部署气流。根据您的环境和所需配置，根据需要修改，添加和 / 或删除 `custom-values.yaml` 文件中的值。

```
$ 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
```

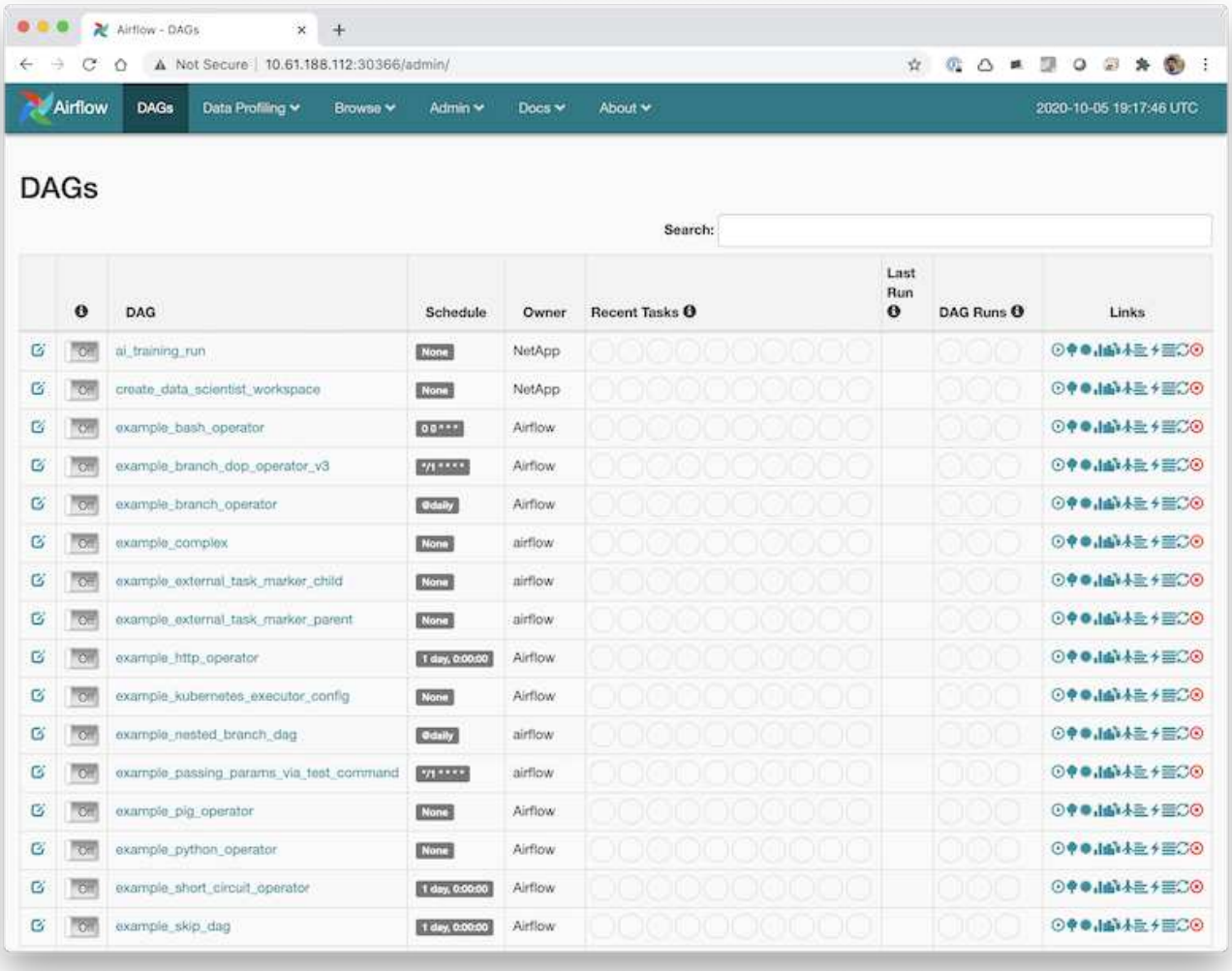
2. 确认所有气流 Pod 均已启动且正在运行。所有 POD 可能需要几分钟的时间才能启动。

```
$ 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 时控制台上印有的说明获取 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。



使用**NetApp**数据操作工具包和**Airflow**

。 "适用于Kubernetes的NetApp DataOps工具包" 可与气流结合使用。通过将NetApp数据操作工具包与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)整合到由Airflow编排的自动化工作流中。

请参见 "气流示例" 部分、NetApp了解有关在Airflow中使用此工具包的详细信息。

Asta三端操作示例

本节包括您可能希望使用Asta三端对其执行的各种操作的示例。

导入现有卷

如果您的 NetApp 存储系统 / 平台上有要挂载到 Kubernetes 集群中的容器上但未与集群中的 PVC 绑定的现有卷，则必须导入这些卷。您可以使用 Trident 卷导入功能导入这些卷。

以下示例命令显示了名为的卷的导入过程 `pb_fg_all`。有关 PVCs 的详细信息，请参见 "Kubernetes 官方文档"。有关卷导入功能的详细信息，请参见 "Trident 文档"。

在示例 PVC 规范文件中指定了 `accessModes` 值 `ReadOnlyMany` 。有关 `accessMode` 字段的详细信息，请参见 "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
```

配置新卷

您可以使用 Trident 在 NetApp 存储系统或平台上配置新卷。

使用**kubec**迎接 新卷的到来

以下示例命令显示了如何使用kubec用 配置新的FlexVol卷。

在以下示例 PVC 定义文件中指定了 `accessModes` 值 `ReadWriteMany` 。有关 `accessMode` 字段的详细信息，请参见 "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**数据操作工具包配置新卷

此外、您还可以使用适用于Kubornetes的NetApp数据操作工具包在NetApp存储系统或平台上配置新卷。适用于Kubnetes的NetApp数据操作工具包利用三端存储来配置卷、但为用户简化了配置过程。请参见 "文档。" 了解详细信息。

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。借助 Trident ，您可以快速轻松地使可能包含数 PB 数据的数据卷可供 Kubernetes 工作负载访问。要使此类数据卷可从 Kubernetes Pod 中访问，只需在 Pod 定义中指定 PVC 即可。

本节假定您已将尝试在 Kubernetes 集群中执行的特定 AI 和 ML 工作负载容器化（采用 Docker 容器格式）。

1. 以下示例命令显示了如何为使用 ImageNet 数据集的 TensorFlow 基准工作负载创建 Kubernetes 作业。有关 ImageNet 数据集的详细信息，请参见 "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，其中包含八个或更多 GPU 的工作节点不存在或当前占用另一个工作负载。如果是，则此作业将保持待定状态，直到此类辅助节点变为可用为止。

   此外，为了最大程度地提高存储带宽，包含所需训练数据的卷会在该作业创建的 POD 中挂载两次。此外，还会在 Pod 中挂载另一个卷。第二个卷将用于存储结果和指标。这些卷在作业定义中使用 PVC 的名称进行引用。有关 Kubernetes 作业的详细信息，请参见 "Kubernetes 官方文档"。

   在本示例作业创建的 Pod 中， `edium` 值为 `Memory` 的` emtyDir `m卷将挂载到 ` `/dev/shm` 。Docker 容器运行时自动创建的 ` /dev/shm` 虚拟卷的默认大小有时可能不足以满足 TensorFlow 的需求。按以下示例所示挂载 `emptyDir` 卷可提供足够大的 ` /dev/shm` 虚拟卷。有关 `emptyDir` 卷的详细信息，请参见 "Kubernetes 官方文档"。

   在此示例作业定义中指定的单个容器将获得 `securityContext > privileged` 值 `true` 。此值表示容器在主机上具有有效的 root 访问权限。在这种情况下使用此标注是因为要执行的特定工作负载需要 root 访问权限。具体而言，工作负载执行的清除缓存操作需要 root 访问权限。是否需要此 **特权：** `true` 标注取决于您要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按照作业定义中的指定为此作业创建了一个 POD ，并且此 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： * 清理作业项目。以下示例命令显示了在步骤 1 中创建的作业对象的删除情况。

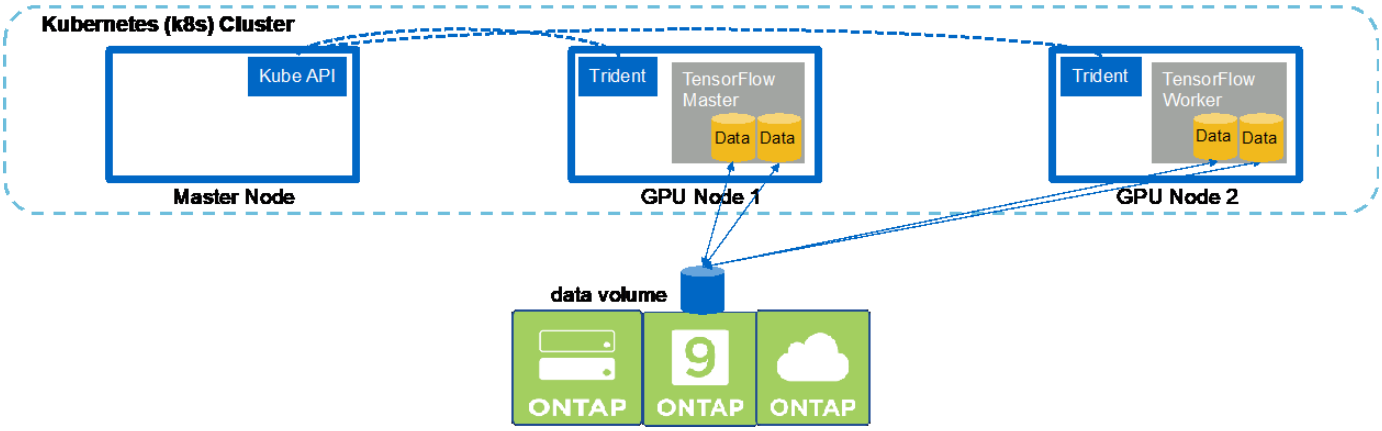
   删除作业对象时， Kubernetes 会自动删除任何关联的 Pod 。

```
$ 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。通过此过程，您可以利用存储在 NetApp 卷上的数据，并使用单个工作节点所能提供的 GPU 。有关同步分布式 AI 作业的描述，请参见下图。

与异步分布式作业相比，同步分布式作业有助于提高性能和训练准确性。本文档不会讨论同步作业与异步作业的利弊。



1. 以下示例命令显示了创建一名员工参与同步分布式执行本节中示例中在单个节点上执行的同一 TensorFlow 基准测试作业的过程 "执行单节点 AI 工作负载"。在此特定示例中，仅部署一个员工，因为此作业会在两个员工节点上执行。

   此示例员工部署请求八个 GPU ，因此可以在一个 GPU 工作节点上运行，该节点具有八个或更多 GPU 。如果您的 GPU 工作节点具有八个以上的 GPU ，则为了最大限度地提高性能，您可能需要增加此数量，使其等于您的工作节点所具有的 GPU 数量。有关 Kubernetes 部署的详细信息，请参见 "Kubernetes 官方文档"。

在此示例中创建了 Kubernetes 部署，因为此特定容器化员工永远不会自行完成。因此，使用 Kubernetes 作业构造来部署它毫无意义。如果员工的设计或编写是为了自己完成，则可以使用此作业构建来部署员工。

在此示例部署规范中指定的 Pod 的值为 `hostNetwork` 值 `true` 。此值表示 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标注是因为特定工作负载依靠 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 的讨论不在本文档的讨论范围之内。是否需要此 `hostNetwork` ： `true` 标注取决于要执行的特定工作负载的要求。有关 `hostNetwork` 字段的详细信息，请参见 "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. 确认您在第 1 步中创建的员工部署已成功启动。以下示例命令确认已为部署创建了一个辅助 POD ，如部署定义所示，并且此 POD 当前正在其中一个 GPU 辅助节点上运行。

```
$ 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. 为启动，参与并跟踪同步多节点作业执行的主节点创建 Kubernetes 作业。以下示例命令创建一个主节点，用于启动，参与和跟踪在一节中的示例中对单个节点执行的相同 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 工作节点具有八个以上的 GPU ，则为了最大限度地提高性能，您可能需要增加此数量，使其等于您的工作节点所具有的 GPU 数量。

   在本示例作业定义中指定的主 Pod 的值为 `hostNetwork` 值为 `true` ，就像在步骤 1 中为工作 Pod 提供了 `hostNetwork` 值 `true` 一样。有关为何需要此值的详细信息，请参见第 1 步。

```
$ 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作业创建了一个主 Pod ，如作业定义所示，并且此 Pod 当前正在其中一个 GPU 工作节点上运行。您还应看到，您最初在步骤 1 中看到的辅助 POD 仍在运行，并且主节点和辅助节点正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. 如果您不再需要此员工部署，请将其删除。以下示例命令显示了删除在步骤 1 中创建的工作部署对象的过程。

   删除 worker 部署对象时， Kubernetes 会自动删除任何关联的 worker Pod 。

```
$ 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： * 清理主作业项目。以下示例命令显示了删除在步骤 3 中创建的主作业对象的过程。

   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.
```

版权信息

商标信息