# 使用 Run AI 优化集群和 GPU 利用率

NetApp Solutions

NetApp
April 12, 2024

# 目录

# 使用 Run： AI 优化集群和 GPU 利用率

以下各节详细介绍了运行： AI 安装，测试场景以及在此验证中执行的结果。

我们使用行业标准基准测试工具（包括 TensorFlow 基准测试）验证了此系统的运行和性能。ImageNet 数据集用于训练 RESNET-50 ，它是一种著名的神经网络（ Convolutional Neural Network ， CNN ） DL 图像分类模型。RESNET-50 可提供准确的训练结果，并缩短处理时间，从而使我们能够对存储产生足够的需求。

## 运行： AI 安装

要安装 Run： AI ，请完成以下步骤：

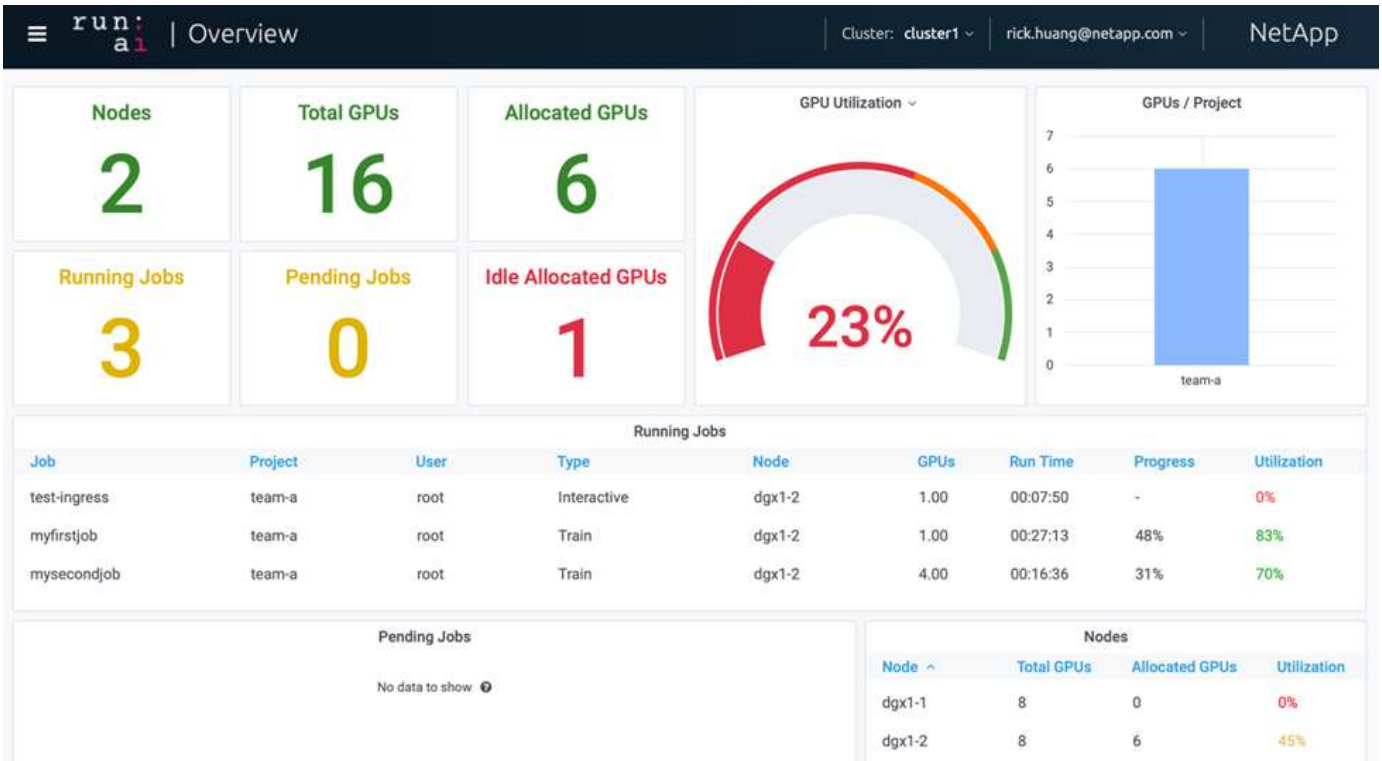
1. 使用 DeepOps 安装 Kubernetes 集群并配置 NetApp 默认存储类。
2. 准备 GPU 节点：
   a. 验证是否已在 GPU 节点上安装 NVIDIA 驱动程序。
   b. 验证是否已安装 `nvidia-Docker` 并将其配置为默认 Docker 运行时。
3. 安装运行： AI ：
   a. 登录到 "运行： AI 管理员 UI" 以创建集群。
   b. 下载创建的 `runai-operator-<clustername>.yaml` 文件。
   c. 将操作员配置应用于 Kubernetes 集群。

   ```
   kubectl apply -f runai-operator-<clustername>.yaml
   ```

4. 验证安装。
   a. 转至 "https://app.run.ai/"。
   b. 转到 " 概述 " 信息板。
   c. 验证右上角的 GPU 数量是否反映了预期的 GPU 数量，并且 GPU 节点均位于服务器列表中。有关 Run： AI 部署的详细信息，请参见 "在内部 Kubernetes 集群上安装 Run： AI" 和 "安装 Run： AI 命令行界面"。

## 运行： AI 信息板和视图

在 Kubernetes 集群上安装 Run： AI 并正确配置容器后，您将看到以下信息板和视图 "https://app.run.ai" 在浏览器中，如下图所示。




集群中共有 16 个 GPU ，由两个 DGX-1 节点提供。您可以查看节点数，可用 GPU 总数，分配给工作负载的已分配 GPU ，正在运行的作业总数，待定作业以及闲置已分配 GPU 。右侧的条形图显示每个项目的 GPU ，其中总结了不同团队如何使用集群资源。中间是当前正在运行的作业列表，其中包含作业详细信息，包括作业名称，项目，用户，作业类型， 每个作业正在运行的节点，为此作业分配的 GPU 数量，作业的当前运行时间，作业进度百分比以及该作业的 GPU 利用率。请注意，集群利用率不足（ GPU 利用率为 23% ），因为一个团队只提交了三个正在运行的作业（`team-A`）。

在下一节中，我们将介绍如何在 " 项目 " 选项卡中创建多个团队，并为每个团队分配 GPU ，以便在每个集群有大量用户时最大限度地提高集群利用率并管理资源。这些测试场景模拟了在训练，推理和交互式工作负载之间共享内存和 GPU 资源的企业环境。

# 为数据科学团队创建项目并分配 GPU

研究人员可以通过 Run： AI CLI， Kubeflow 或类似流程提交工作负载。为了简化资源分配并创建优先级， Run： AI 引入了项目概念。项目是指将项目名称与 GPU 分配和首选项关联的配额实体。这是一种管理多个数据科学团队的简单便捷的方式。

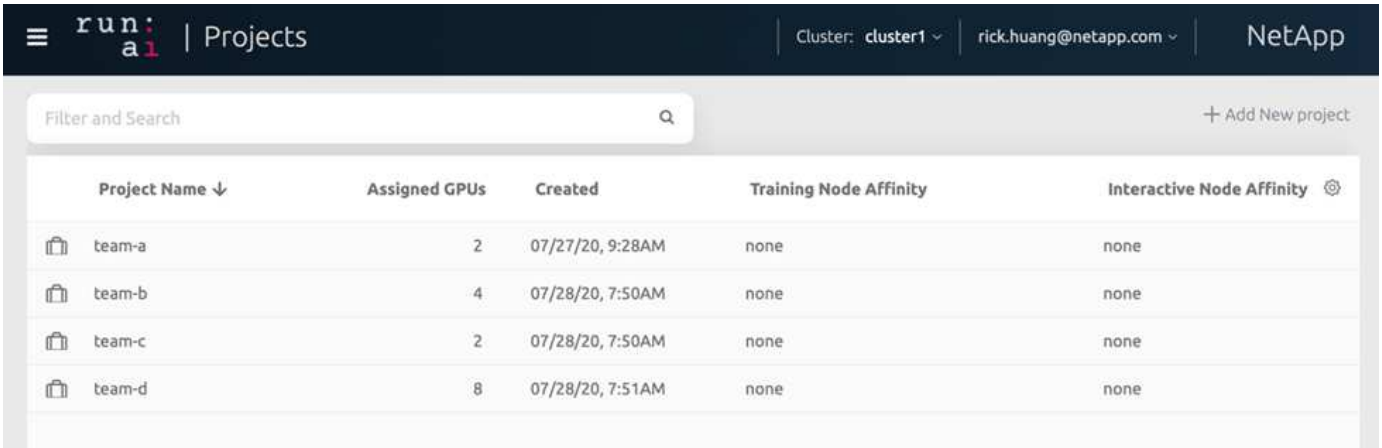
提交工作负载的研究人员必须将项目与工作负载请求相关联。运行： AI 计划程序会将请求与当前分配和项目进行比较，并确定是否可以为工作负载分配资源或是否应保持待定状态。

作为系统管理员，您可以在 Run： AI projects 选项卡中设置以下参数：

- * 模拟项目。 * 为每个用户设置一个项目，为每个用户团队设置一个项目，并为每个实际组织项目设置一个项目。
- * 项目配额。 * 每个项目都与一个 GPU 配额相关联，可以同时为此项目分配 GPU 配额。这是一个有保障的配额，因为使用此项目的研究人员无论在集群中处于何种状态，都可以获得这一数量的 GPU 。通常，项目分配总和应等于集群中的 GPU 数量。除此之外，此项目的用户还可以收到超配额。只要未使用 GPU ，使用此项目的研究人员就可以获得更多 GPU 。我们在中演示了超额配额测试场景和公平考虑事项 "通过过度配额 GPU 分配实现高集群利用率"， "基本资源分配公平"，和 "配额过度公平"。

- 创建新项目，更新现有项目并删除现有项目。
- * 限制作业在特定节点组上运行 * 。您可以分配仅在特定节点上运行的特定项目。如果项目团队需要专用硬件，例如具有足够内存的硬件，则此功能非常有用。或者，项目团队也可能是特定硬件的所有者，这些硬件是通过专门预算获得的，或者您可能需要直接构建或交互式工作负载来处理较弱的硬件，并将较长的培训或无人看管的工作负载定向到较快的节点。有关对节点进行分组并为特定项目设置相关性的命令，请参见 "运行： AI 文档"。
- * 限制交互作业的持续时间 * 。研究人员经常忘记关闭交互式作业。这可能会导致资源浪费。一些组织倾向于限制交互式作业的持续时间并自动关闭这些作业。

下图显示了已创建四个团队的 " 项目 " 视图。每个团队分配不同数量的 GPU 来处理不同的工作负载， GPU 总数等于由两个 DGX-1 组成的集群中可用 GPU 总数。




# 在 Run： AI 命令行界面中提交作业

本节详细介绍了可用于运行任何 Kubernetes 作业的基本 Run： AI 命令。它会根据工作负载类型分为三部分。AI/ML/DL 工作负载可分为两种通用类型：

- * 无人参与的培训课程 * 。对于这些类型的工作负载，数据科学家会准备一个自运行的工作负载并将其发送给执行。执行期间，客户可以检查结果。此类工作负载通常用于生产或模型开发阶段，无需人工干预。
- * 交互式构建会话 * 。对于这些类型的工作负载，数据科学家将与 Bash ， Jupyter Notebook ，远程 PyCharm 或类似的 IDE 打开交互式会话，并直接访问 GPU 资源。我们还提供了第三种使用连接的端口运行交互式工作负载的方案，以便向容器用户显示内部端口。

## 无人参与的培训工作负载

设置项目并分配 GPU 后，您可以在命令行中使用以下命令运行任何 Kubernetes 工作负载：

```
$ runai project set team-a runai submit hyper1 -i gcr.io/run-ai-
demo/quickstart -g 1
```

此命令将为团队 A 启动无人参与的培训作业，并分配一个 GPU 。此作业基于示例 Docker 映像 `gcr.io/run-ai-demo/Quickstart` 。我们将作业命名为 `hyper1` 。然后，您可以运行以下命令来监控作业的进度：

```
$ runai list
```

下图显示了 `runai list` 命令的结果。您可能看到的典型状态包括：

- **容器创建** 。正在从云存储库下载 Docker 容器。
- **待定** 。作业正在等待计划。
- **运行** 。作业正在运行。

```
~> runai list
Showing jobs for project team-a
NAME    STATUS   AGE  NODE                                    IMAGE                        TYPE        PROJECT  USER   GPUs
hyper1  Running  11s  gke-dev-yaron1-gpu-4-pool-154f511d-5nk5  gcr.io/run-ai-demo/quickstart  Train       team-a   yaron  1
```

要获取作业的其他状态，请运行以下命令：

```
$ runai get hyper1
```

要查看作业日志，请运行 `runai logs <job-name>` 命令：

```
$ runai logs hyper1
```

在此示例中，您应看到正在运行的 DL 会话的日志，包括当前训练时间， ETA ，损失函数值，准确性以及每个步骤所用时间。

您可以在运行： AI UI 上查看集群状态，网址为 "https://app.run.ai/"。在 Dashboards > Overview 下，您可以监控 GPU 利用率。

要停止此工作负载，请运行以下命令：

```
$ runai delte hyper1
```

此命令将停止训练工作负载。您可以再次运行 `runai list` 来验证此操作。有关详细信息，请参见 "启动无人参与的培训工作负载"。

## 交互式构建工作负载

设置项目并分配 GPU 后，您可以在命令行中使用以下命令运行交互式构建工作负载：

```
$ runai submit build1 -i python -g 1 --interactive --command sleep --args
infinity
```

此作业基于示例 Docker 映像 python 。我们将作业 BUILD1 命名为。

` - 交互式` 标志表示作业没有开始或结束研究人员有责任完成此项工作。管理员可以为交互式作业定义一个时间限制，在该时间限制之后，系统会终止这些作业。

` -g 1` 标志可为此作业分配一个 GPU 。提供的命令和参数为 ` —命令休眠— args infinity` 。您必须提供命令，否则容器将立即启动并退出。

以下命令的工作方式与中所述的命令类似 [无人参与的培训工作负载]：

- `runai list` ：显示名称，状态，期限，节点，映像， 用于作业的项目，用户和 GPU 。
- `runai get build1` ：显示作业 build1 的其他状态。
- `runai delete build1` ：停止交互式工作负载 BUILD1.To get a bash shell to the container ， the following command ：

```
$ runai bash build1
```

这样就可以直接将 shell 连接到计算机。然后，数据科学家可以在容器中开发或微调其模型。

您可以在运行： AI UI 上查看集群状态，网址为 "https://app.run.ai"。有关详细信息，请参见 "启动和使用交互式构建工作负载"。

### 使用已连接端口的交互式工作负载

作为交互式构建工作负载的扩展，在使用 Run ： AI CLI 启动容器时，您可以向容器用户显示内部端口。这对于云环境，使用 Jupyter 笔记本电脑或连接到其他微服务非常有用。 "传入" 允许从 Kubernetes 集群外部访问 Kubernetes 服务。您可以通过创建一组规则来配置访问，这些规则定义哪些入站连接访问哪些服务。

为了更好地管理对集群中服务的外部访问，我们建议集群管理员安装 "传入" 并配置负载平衡器。

要使用传入作为服务类型，请在提交工作负载时运行以下命令以设置方法类型和端口：

```
$ runai submit test-ingress -i jupyter/base-notebook -g 1 \
  --interactive --service-type=ingress --port 8888 \
  --args="--NotebookApp.base_url=test-ingress" --command=start-notebook.sh
```

容器成功启动后，执行 `runai list` 以查看用于访问 Jupyter 笔记本电脑的 `S`服务 `URL` （ `S` ）。此 URL 由入口端点，作业名称和端口组成。例如，请参见 https://10.255.174.13/test-ingress-8888。

有关详细信息，请参见 "使用连接的端口启动交互式构建工作负载"。

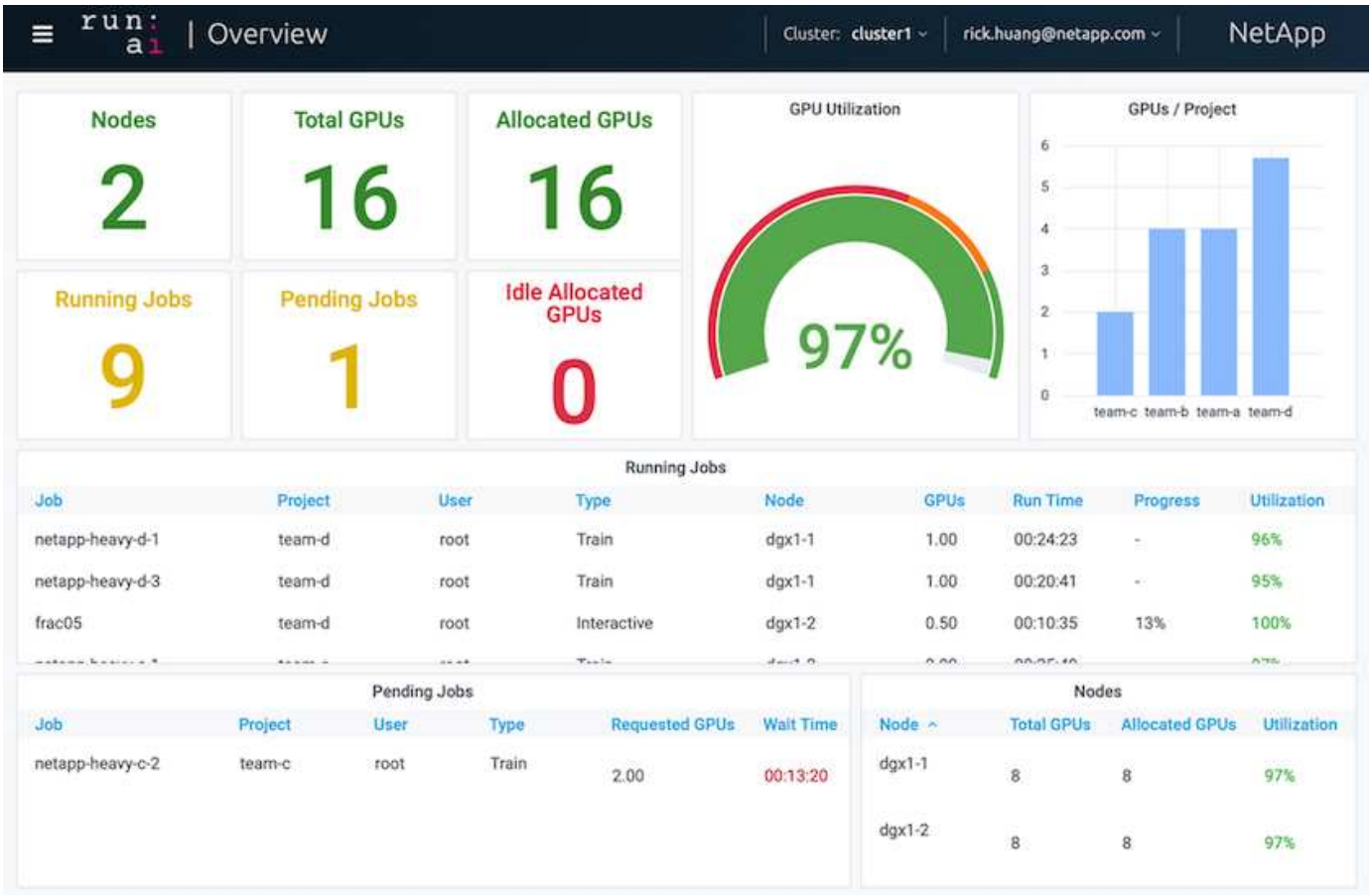
## 实现高集群利用率

在本节中，我们模拟了一个实际场景，其中四个数据科学团队各自提交自己的工作负载，以展示 Run ： AI Orchestration 解决方案，它可以在保持 GPU 资源优先级和平衡的同时实现高集群利用率。我们首先使用一节中所述的 RESNET-50 基准测试 "使用 ImageNet 数据集的 RESNET-50 基准测试摘要"：

```
$ runai submit netapp1 -i netapp/tensorflow-tf1-py3:20.01.0 --local-image
--large-shm  -v /mnt:/mnt -v /tmp:/tmp --command python --args
"/netapp/scripts/run.py" --args "--
dataset_dir=/mnt/mount_0/dataset/imagenet/imagenet_original/" --args "--
num_mounts=2"  --args "--dgx_version=dgx1" --args "--num_devices=1" -g 1
```

我们运行的 RESNET-50 基准测试与中相同 "NVA-1121"。对于不驻留在公有 Docker 存储库中的容器，我们使用了标志 ` -local-image` 。我们将主机 DGX-1 节点上的目录 ` /mnt` 和 ` /tm` 分别挂载到 ` /mnt` 和 ` /tm` 。该数据集位于 NetApp AFFA800 上，并且 `dataset_dir` 参数指向目录。` -num_devices=1` 和 ` -g 1` 表示我们为此作业分配一个 GPU 。前者是 `run.py` 脚本的参数，而后者是 `runai Submit` 命令的标志。

下图显示了一个系统概述信息板，其中 GPU 利用率为 97% ，所有十六个可用 GPU 均已分配。您可以在 GPU/项目条形图中轻松查看为每个团队分配的 GPU 数量。" 正在运行的作业 " 窗格显示当前正在运行的作业名称，项目，用户，类型，节点， GPU 已用，运行时间，进度和利用率详细信息。队列中的工作负载列表及其等待时间显示在 "Pending" 作业中。最后，节点框将提供集群中各个 DGX-1 节点的 GPU 编号和利用率。




## 为要求较低的工作负载或交互式工作负载分配的 GPU 百分比

当研究人员和开发人员在开发，超参数调整或调试阶段使用其模型时，此类工作负载通常所需的计算资源更少。因此，配置百分比 GPU 和内存的效率更高，以便可以将同一 GPU 同时分配给其他工作负载。Run ： AI 的业务流程解决方案为 Kubernetes 上的容器化工作负载提供了一个百分比 GPU 共享系统。该系统支持运行 CUDA 程序的工作负载，尤其适

用于推理和模型构建等轻型 AI 任务。部分 GPU 系统可以透明地为数据科学和 AI 工程团队提供在一个 GPU 上同时运行多个工作负载的能力。这样，企业就可以在同一硬件上运行更多的工作负载，例如计算机视觉，语音识别和自然语言处理，从而降低成本。

Run： AI 的百分比 GPU 系统可利用自身的内存和计算空间有效地创建虚拟化逻辑 GPU ，容器可以使用和访问这些 GPU ，就像它们是独立的处理器一样。这样，多个工作负载便可在同一 GPU 上的容器中并排运行，而不会相互干扰。解决方案是透明，简单且可移植的，不需要对容器本身进行更改。

一个典型的使用情形可能会看到在同一个 GPU 上运行两到八个作业，这意味着您可以使用同一个硬件执行八倍的工作。

对于下图中的作业 `frac05` 属于项目 `team-d` ，我们可以看到分配的 GPU 数量为 0.5 。这一点可通过 `nvidia-smi` 命令进一步验证，该命令显示容器可用的 GPU 内存为 16 ， 255 MB： DGX-1 节点中每个 V100 GPU 32 GB 的一半。

```
root@run-deploy:~# runai bash frac05 -p team-d
root@frac05-0:/workload# nvidia-smi
Tue Jul 28 15:17:03 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.05    Driver Version: 450.51.05    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   57C    P0   240W / 300W |  15525MiB / 16255MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       156      C   python3                         15525MiB |
+-----------------------------------------------------------------------------+
```

## 通过过度配额 GPU 分配实现高集群利用率

在本节和各节中 "基本资源分配公平"，和 "配额过度公平"，我们设计了高级测试方案，用于演示运行： AI 流程编排功能，以实现复杂的工作负载管理，自动抢占式计划和超配额 GPU 配置。我们这样做是为了在 ONTAP AI 环境中实现高集群资源利用率并优化企业级数据科学团队的工作效率。

对于这三个部分，请设置以下项目和配额：

| 项目 | 配额 |
| --- | --- |
| 团队 A | 4. |
| 团队 b | 2. |
| 团队 c | 2. |
| 团队 | 8. |

此外，我们还会对这三个部分使用以下容器：

- Jupyter 笔记本电脑： `jupyter/base-notebook`
- Run ： AI Quickstart ： `gcr.io/run-ai-demo/Quickstart`

我们为此测试场景设定了以下目标：

- 展示资源配置的简便性以及如何从用户中提取资源
- 展示用户如何轻松配置 GPU 的小部分和 GPU 的整数
- 展示如果集群中存在可用 GPU ，系统如何通过允许团队或用户超过其资源配额来消除计算瓶颈
- 展示如何在运行计算密集型作业（例如 NetApp 容器）时使用 NetApp 解决方案消除数据管道瓶颈
- 显示如何使用系统运行多种类型的容器
  - Jupyter 笔记本电脑
  - 运行： AI 容器
- 集群已满时显示高利用率

有关在测试期间执行的实际命令序列的详细信息，请参见 "第 4.8 节的测试详细信息"。

提交所有 13 个工作负载后，您可以看到一个容器名称和分配的 GPU 列表，如下图所示。我们有七个培训和六个互动作业，模拟四个数据科学团队，每个团队都有自己的模型运行或开发。对于交互式作业，各个开发人员都在使用 Jupyter 笔记本电脑编写或调试其代码。因此，它适合在不使用过多集群资源的情况下配置 GPU 分段。

```
root@run-deploy:~# runai list -A
NAME           STATUS   AGE  NODE    IMAGE                          TYPE        PROJECT  USER  GPUs  CREATED BY CLI  SERVICE URL(S)
b-4-gg         Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-b   root  2     true
c-5-g          Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-c   root  1     true
c-4-gg         Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-c   root  2     true
b-3-g          Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-b   root  1     true
c-3-g02        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.2   true
d-1-gggg       Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-d   root  4     true
c-2-g03        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.3   true
c-1-g05        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.5   true
a-2-gg         Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-a   root  2     true
b-2-g04        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive team-b   root  0.4   true
a-1-g          Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-a   root  1     true
b-1-g06        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive team-b   root  0.6   true
a-1-1-jupyter  Running  3m   dgx1-1  jupyter/base-notebook          Interactive team-a   root  1     true            http://10.61.218.134/a-1-1-jupyter,
https://10.61.218.134/a-1-1-jupyter
```

此测试场景的结果如下：

- 集群应已满：使用了 16/16 个 GPU 。
- 集群利用率高。
- 由于分配百分比的影响，比 GPU 的实验更多。

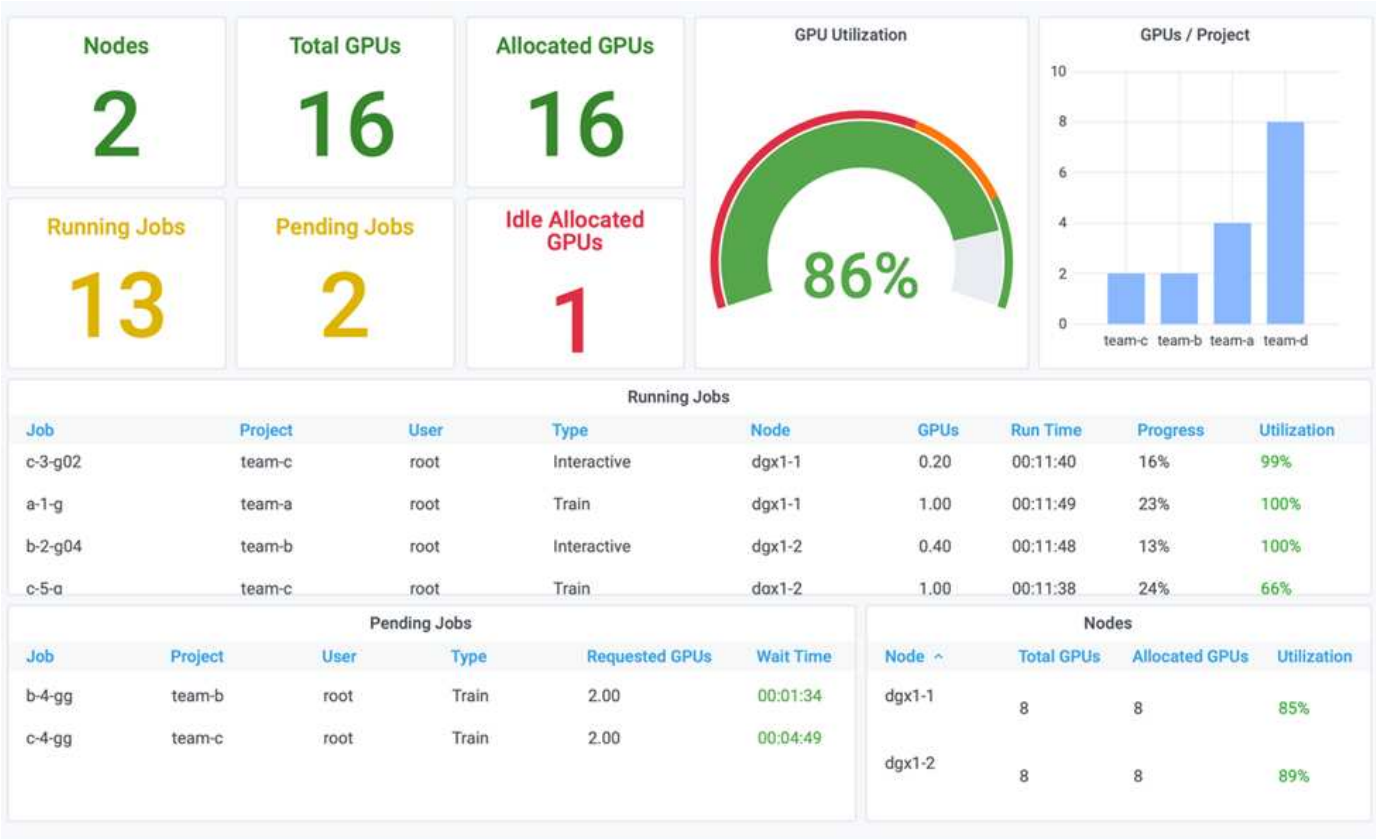
- team-d 并未使用所有配额；因此， team-b 和 team-c 可以在其实验中使用额外的 GPU ，从而加快创新速度。

# 基本资源分配公平

## 在本节中，我们会显示，当 team-d 请求更多 GPU （它们低于其配额）时，系统会暂停 team-b 和 team-c 的工作负载，并以公平的方式将其移至待定状态。

有关提交作业，使用的容器映像以及执行的命令序列等详细信息，请参见一节 "第 4.9 节的测试详细信息"。

下图显示了由于自动负载平衡和预先计划而产生的集群利用率，每个组分配的 GPU 以及待处理作业。我们可以观察到，当所有团队工作负载请求的 GPU 总数超过集群中可用的 GPU 总数时， Run： AI 的内部公平算法会分别为 team-b 和 team-c 暂停一个作业，因为它们已达到项目配额。这样可以提供整体较高的集群利用率，而数据科学团队仍在管理员设置的资源限制下工作。




此测试场景的结果显示以下内容：

- * 自动负载平衡。 * 系统会自动平衡 GPU 的配额，使每个团队现在都在使用其配额。暂停的工作负载属于超过其配额的团队。
- * 公平共享暂停。 * 系统会选择停止超过配额的一个组的工作负载，然后停止另一个组的工作负载。Run： AI 具有内部公平算法。

# 配额过度公平

## 在本节中，我们将扩展多个团队提交工作负载并超过其配额的情形。通过这种方式，我们

## 将展示 Run ： AI 的公平性算法如何根据预设配额比率分配集群资源。

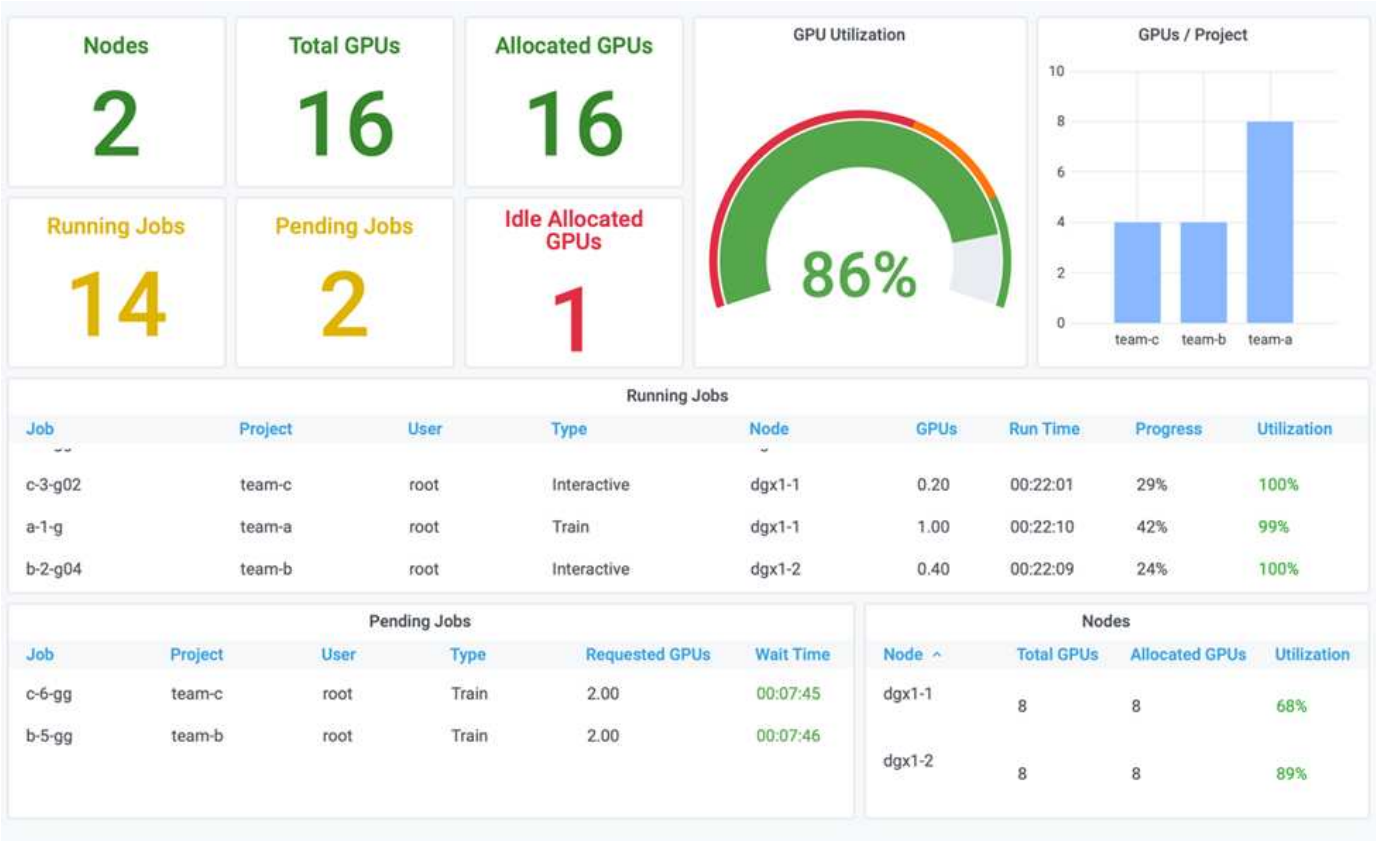
此测试场景的目标：

- 显示多个团队请求超过其配额的 GPU 时的排队机制。
- 显示系统如何根据配额之间的比率在超过配额的多个组之间分配公平的集群份额，以便具有较大配额的组获得较大的备用容量份额。

结束时 "基本资源分配公平"，有两个工作负载排队：一个用于 `team-b` ，一个用于 `team-c` 。在本节中，我们将对其他工作负载进行排队。

有关提交作业，使用的容器映像以及执行的命令序列等详细信息，请参见 "第 4.10 节的测试详细信息"。

根据部分提交所有作业时 "第 4.10 节的测试详细信息"系统信息板显示 `team-A` ， `team-b` 和 `team-c` 所有 GPU 都超过其预设配额。与预设的软配额（4 个）相比， `team-A` 占用的 GPU 更多，而 `team-b` 和 `team-c` 每个占用的 GPU 都比其软配额（2 个）多两个。分配的过度配额 GPU 的比率等于其预设配额的比率。这是因为系统使用预设配额作为优先级的参考，并在多个团队请求更多 GPU ，超过其配额时相应地进行配置。当企业数据科学团队积极参与 AI 模型的开发和生产时，这种自动负载平衡可以实现公平和优先级划分。



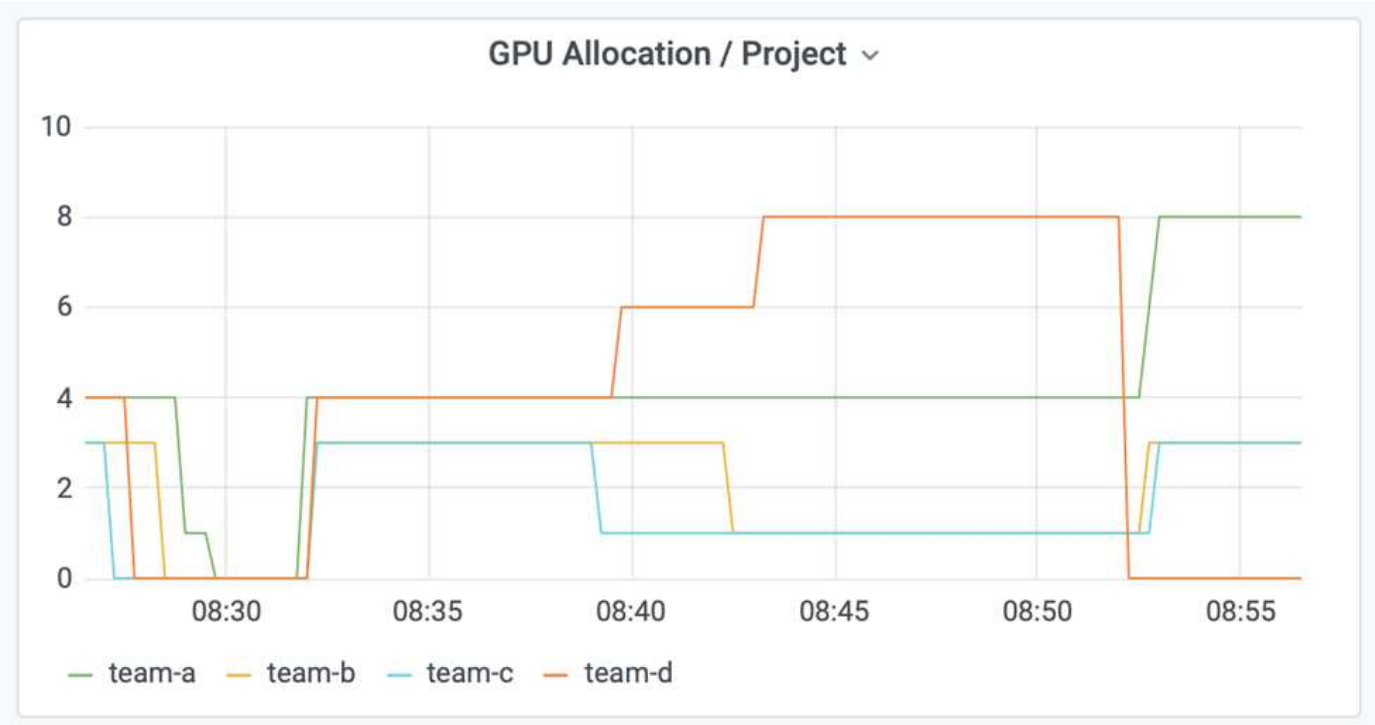

此测试场景的结果如下：

- 系统开始将其他团队的工作负载取消队列。
- 除队顺序根据公平算法来确定，这样 `team-b` 和 `team-c` 会获得相同数量的超配额 GPU （因为它们具有类似的配额）， 而 `team-A` 获得的 GPU 数量是原来的两倍，因为他们的配额是 `team-b` 和 `team-c` 的两倍。
- 所有分配都将自动完成。

因此，系统应在以下状态下保持稳定：

| 项目 | 已分配 **GPU** | **comment** |
| --- | --- | --- |
| 团队 A | 8/4. | 超过配额的四个 GPU 。空队列。 |
| 团队 b | 4/2 | 超过配额的两个 GPU 。一个工作负载已排队。 |
| 团队 c | 4/2 | 超过配额的两个 GPU 。一个工作负载已排队。 |
| 团队 | 0/8 | 根本不使用 GPU ，没有已排队的工作负载。 |

下图显示了各个部分的 Run ： AI Analytics 信息板中每个项目在一段时间内的 GPU 分配情况 "通过过度配额 GPU 分配实现高集群利用率"， "基本资源分配公平"，和 "配额过度公平"。图中的每一行表示在任何时间为给定数据科学团队配置的 GPU 数量。我们可以看到，系统会根据提交的工作负载动态分配 GPU 。这样，当集群中存在可用 GPU 时，团队可以超过配额，然后根据公平原则抢占作业，最后达到所有四个团队的稳定状态。




# 将数据保存到 Trident 配置的 PersistentVolume

NetApp Trident 是一个完全受支持的开源项目，旨在帮助您满足容器化应用程序的复杂持久性需求。您可以将数据读写到 Trident 配置的 Kubernetes PersistentVolume （ PV ）中，并通过 NetApp ONTAP 数据管理软件提供数据分层，加密， NetApp Snapshot 技术，合规性和高性能优势。

## 重复使用现有命名空间中的 PVC

对于规模较大的 AI 项目，不同容器向同一个 Kubernetes PV 读取和写入数据可能会更高效。要重复使用 Kubernetes 永久性卷声明（ PVC ），用户必须已创建 PVC 。请参见 "NetApp Trident 文档" 有关创建 PVC 的

详细信息。以下是重复使用现有 PVC 的示例：

```
$ runai submit pvc-test -p team-a --pvc test:/tmp/pvc1mount -i gcr.io/run-ai-demo/quickstart -g 1
```

运行以下命令查看项目 `team-A` 的作业 `pvc` 测试 的状态：

```
$ runai get pvc-test -p team-a
```

您应看到 PV /tmp/pvc1mount 挂载到 `team-A` job `vc-test` 。这样，多个容器就可以从同一个卷读取数据，这在开发或生产环境中存在多个竞争模式时非常有用。数据科学家可以构建一系列模型，然后通过多数投票或其他技术将预测结果结合起来。

使用以下命令访问容器 Shell：

```
$ runai bash pvc-test -p team-a
```

然后，您可以检查已挂载的卷并访问容器中的数据。

这种重复使用 PVC 的功能可与 NetApp FlexVol 卷和 NetApp ONTAP FlexGroup 卷配合使用，从而使数据工程师可以使用更灵活，更强大的数据管理选项来利用由 NetApp 提供支持的数据网络结构。