実行： AI を使用したネットアップオーケストレーション解決策

NetApp Solutions

NetApp
April 10, 2024

目次

TR-4858 ： 『 NetApp Orchestration 解決策 with Run ： AI 』

ネットアップの Yaron Goldberg 、 Run ： AI 、 David Arnette 、 Sung-Han Lin

NetApp AFF ストレージシステムは、卓越したパフォーマンスと業界をリードするハイブリッドクラウドデータ管理機能を提供します。ネットアップと Run ： AI は、 NetApp ONTAP AI 解決策の独自の機能である人工知能（ AI ）と機械学習（ ML ）のワークロードに対応し、エンタープライズクラスのパフォーマンス、信頼性、サポートを提供していることを実証するために提携しました。実行： AI ワークロードの AI オーケストレーションにより、 Kubernetes ベースのスケジュールとリソース利用プラットフォームが追加され、研究者が GPU 利用率を管理、最適化できるようになります。ネットアップ、 NVIDIA 、 Run ： AI の解決策を組み合わせた NVIDIA DGX システムは、エンタープライズ AI ワークロードに特化したインフラスタックを提供します。このテクニカルレポートでは、さまざまなユースケースや業種に対応した会話型 AI システムを構築するお客様向けのガイダンスを提供します。Run の導入に関する情報、 AI と NetApp AFF A800 ストレージシステムが記載されています。 AI イニシアチブを短期間で成功に導くためのリファレンスアーキテクチャとして機能します。

解決策の対象となるグループは次のとおりです。

- AI 開発のためのソリューションを設計するエンタープライズアーキテクト コンテナ化などの Kubernetes ベースのユースケース向けのモデルとソフトウェア マイクロサービス
- データサイエンティストは、効率的なモデルを実現するための効率的な方法を探しています 複数のチームとで構成されるクラスタ環境における開発目標 プロジェクト
- 本番モデルの保守と実行を担当するデータエンジニア
- エグゼクティブや IT の意思決定者、ビジネスリーダー Kubernetes クラスタのリソース利用率を最適化する方法をご確認ください AI 導入の市場投入までの時間を短縮できます

解決策の概要

NetApp ONTAP AI と AI のコントロールプレーン

ネットアップと NVIDIA が開発、検証した NetApp ONTAP AI アーキテクチャは、 NVIDIA DGX システムとネットアップのクラウド対応ストレージシステムを基盤としています。このリファレンスアーキテクチャには、 IT 組織に次のようなメリットがあります。

- 設計の複雑さを解消
- コンピューティングとストレージを個別に拡張できます
- 小規模構成から始めて、シームレスに拡張できます
- は、さまざまなパフォーマンスとに対応するさまざまなストレージオプションを提供します コストポイント

NetApp ONTAP AI は、 DGX システムと NetApp AFF A800 ストレージシステムを最先端のネットワークと緊

密に統合します。NetApp ONTAP AI システムと DGX システムでは、設計の複雑さと推測に頼らず、 AI 導入を簡易化できます。お客様は小規模構成から始めて、システムを中断なく拡張できます。同時に、エッジ、コア、クラウドにわたってデータをインテリジェントに管理できます。

ネットアップの AI コントロールプレーンは、データサイエンティストとデータエンジニアを対象とした、フルスタックの AI 、 ML 、ディープラーニング（ DL ）のデータと実験管理解決策です。AI の利用が拡大するにつれて、ワークロードの拡張性やデータの可用性など、さまざまな課題が生じています。NetApp AI コントロールプレーンは、データネームスペースの迅速なクローニングを Git レポジトリと同様に行う機能や、トレーサビリティとバージョン管理のためにデータやモデルベースラインをほぼ瞬時に作成する AI トレーニングワークフローを定義して実装するなど、これらの課題に対処します。NetApp AI コントロールプレーンを使用すると、サイトやリージョン間でデータをシームレスにレプリケートし、 Jupyter Notebook ワークスペースをすばやくプロビジョニングして、大規模なデータセットにアクセスできます。

実行： **AI** ワークロードのオーケストレーション向けの **AI** プラットフォーム

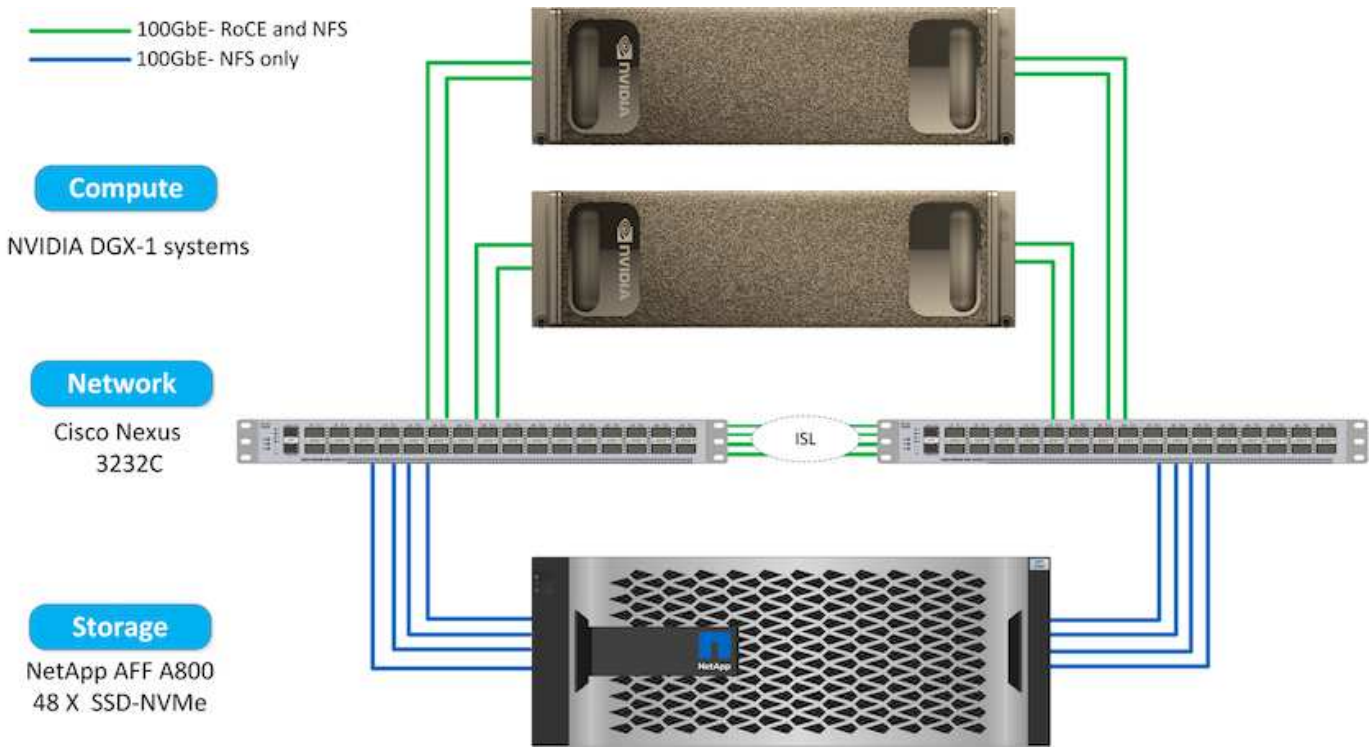
実行： AI は、世界初の AI インフラストラクチャのオーケストレーションおよび仮想化プラットフォームを構築しました。基盤となるハードウェアからワークロードを抽象化することで、 Run ： AI は、 GPU リソースの共有プールを作成します。この共有プールを動的にプロビジョニングし、 AI ワークロードの効率的なオーケストレーションと GPU の使用の最適化を実現します。データサイエンティストは、大量の GPU パワーをシームレスに消費して研究を向上させ、加速させることができます。一方、 IT チームは、リソースのプロビジョニング、キューイング、利用率に関する一元化されたクロスサイト管理とリアルタイムの可視性を維持します。Run ： AI プラットフォームは Kubernetes を基盤として構築されているため、既存の IT ワークフローやデータサイエンスワークフローと簡単に統合できます。

Run ： AI プラットフォームには、次のようなメリットがあります。

- * イノベーションにかかる時間を短縮 * Run ： AI リソースのプール化、キューイング、優先付けのメカニズムをネットアップストレージシステムと組み合わせることで、研究者たちはインフラ管理の苦労から取り取り除かれ、データサイエンスに専念することができます。実行： AI とネットアップのお客様は、コンピューティングやデータパイプラインのボトルネックを発生させることなく、必要な数のワークロードを実行することで生産性を向上できます。
- * チームの生産性向上。 * 実行： AI 公正性アルゴリズムにより、すべてのユーザーとチームが公平なリソースを共有できるようになります。優先度の高いプロジェクトを中心としたポリシーをあらかじめ設定しておくことで、あるユーザやチームから別のユーザやチームにリソースを動的に割り当てることができ、誰もが切望した GPU リソースにタイムリーにアクセスできるようになります。
- * GPU 利用率の向上。 * 実行： AI スケジューラを使用すると、 Kubernetes での分散トレーニング用に、フラクショナルな GPU 、整数型 GPU 、複数の GPU ノードを簡単に利用できます。このように、 AI ワークロードは容量ではなくニーズに基づいて実行されます。データサイエンスチームは、 1 つのインフラでより多くの AI 実験を実行できるようになります。

解決策テクノロジ

この解決策は、 1 台の NetApp AFF A800 システム、 2 台の DGX-1 サーバ、 2 台の Cisco Nexus 3232C 100GbE スイッチで実装されました。DGX-1 サーバはそれぞれ 4 本の 100GbE 接続で Nexus スイッチに接続されます。この接続は、 Converged Ethernet （ RoCE ）を介したリモートダイレクトメモリアクセス（ RDMA ）を使用する GPU 間通信に使用されます。NFS ストレージアクセス用の従来の IP 通信も、これらのリンクで行われます。各ストレージコントローラは、 4 つの 100GbE リンクを使用してネットワークスイッチに接続されています。次の図に、このテクニカルレポートですべてのテストシナリオに使用した ONTAP AI 解決策アーキテクチャを示します。



この解決策で使用されているハードウェア

この解決策は、 ONTAP AI リファレンスアーキテクチャ DGX 1 ノードと AFF A800 ストレージシステム 1 台を使用して検証されました。を参照してください "NVA-1121." を参照してください。

次の表に、テストで解決策を実装するために必要なハードウェアコンポーネントを示します。

ハードウェア	数量
DGX-1 システム	2.
AFF A800	1.
Nexus 3232C スイッチ	2.

ソフトウェア要件

この解決策は、 Run ： AI オペレータがインストールされた基本的な Kubernetes 環境で検証されました。Kubernetes はを使用して導入されました "NVIDIA DeepOps のことです" 導入エンジン：本番環境に必要なすべてのコンポーネントを導入します。DeepOps は自動的に導入されます "NetApp Trident" Kubernetes 環境との永続的なストレージ統合のために、デフォルトのストレージクラスが作成されました。これにより、コンテナは AFF A800 ストレージシステムのストレージを活用できます。Trident と ONTAP AI の Kubernetes の詳細については、を参照してください "TR-4798"。

次の表に、テストで解決策を実装するために必要なソフトウェアコンポーネントを示します。

ソフトウェア	バージョンまたはその他の情報
NetApp ONTAP データ管理ソフトウェア	9.6p4
Cisco NX-OS スイッチのファームウェア	7.0 （3） I6 （1）

ソフトウェア	バージョンまたはその他の情報
NVIDIA DGX OS	4.0.4 - Ubuntu 18.04 LTS
Kubernetes のバージョン	1.17
Trident のバージョン	20.04.0
AI CLI を実行	v2.1.13
実行： AI Orchestration Kubernetes Operator バージョン	1.0.39
Docker コンテナプラットフォーム	18.06.1-CE [e68fc7a]

実行に必要なその他のソフトウェア要件： AI は、から入手できます "AI GPU クラスタの前提条件を実行します"。

クラスタと GPU の利用率を最適化して実行： AI

以下のセクションでは、この検証で実行した AI のインストール、テストシナリオ、結果について詳しく説明します。

TensorFlow ベンチマークを含む業界標準のベンチマークツールを使用して、このシステムの動作とパフォーマンスを検証しました。ImageNet データセットを使用して ResNet-50 をトレーニングしました。これは、画像分類のための有名な Convolutional Neural Network （ CNN ；畳み込みニューラルネットワーク） DL モデルです。ResNet-50 は、処理時間を短縮して正確なトレーニング結果を提供します。これにより、ストレージに対する十分な需要を喚起することができました。

「 AI Installation 」を実行します

Run ： AI をインストールするには、次の手順を実行します。

1. DeepOps を使用して Kubernetes クラスタをインストールし、ネットアップのデフォルトストレージクラスを設定します。
2. GPU ノードの準備：
   a. NVIDIA ドライバが GPU ノードにインストールされていることを確認します。
   b. 「 nvidia - Docker 」がインストールされ、デフォルトの Docker ランタイムとして設定されていることを確認します。
3. インストール実行： AI ：
   a. にログインします "AI 管理 UI を実行" クラスタを作成できます。
   b. 作成した「 runai-operator-<clustername> .yaml 」ファイルをダウンロードします。
   c. オペレータ設定を Kubernetes クラスタに適用します。

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

4. インストールを確認します。

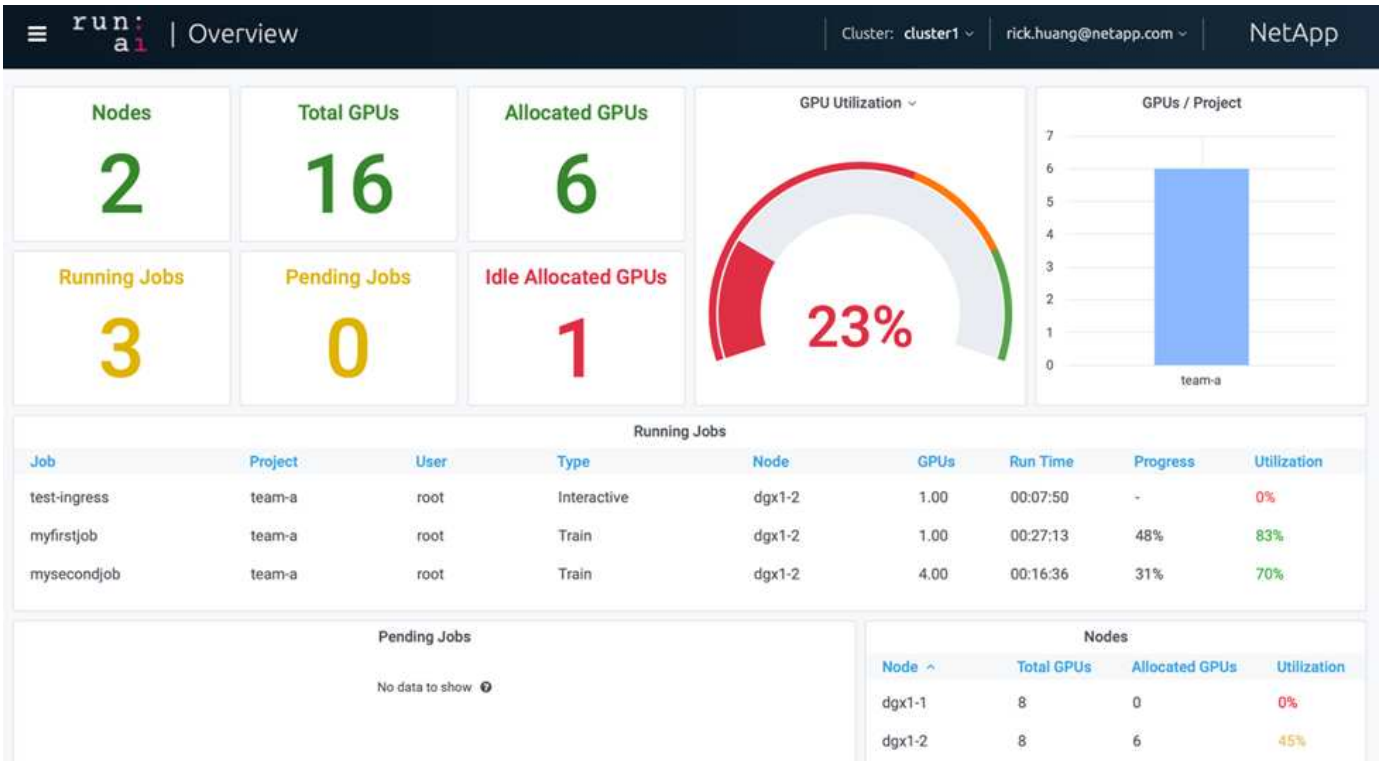
a. に進みます "https://app.run.ai/"。

b. 概要ダッシュボードに移動します。

c. 右上の GPU の数が、想定される GPU の数と GPU ノードの数をすべてサーバのリストに表示していることを確認します。実行： AI 導入の詳細については、を参照してください "Run ： AI をオンプレミスの Kubernetes クラスタにインストール" および "Run ： AI CLI のインストール"。

「 AI Dashboards and Views 」を実行します

Run ： AI を Kubernetes クラスタにインストールし、コンテナを正しく設定したら、次のダッシュボードとビューが表示されます "https://app.run.ai" 次の図に示すように、ブラウザで設定します。



2 つの DGX-1 ノードによってクラスタ内に提供される GPU は合計 16 個です。ノード数、使用可能な GPU の総数、ワークロードに割り当てられている GPU の数、実行中のジョブの総数、保留中のジョブ、およびアイドル状態に割り当てられている GPU の数を確認できます。右側のバー図は、プロジェクトごとの GPU を示しています。これは、各チームがどのようにクラスタリソースを使用しているかをまとめたものです。中央には、ジョブ名、プロジェクト、ユーザー、ジョブタイプなど、現在実行中のジョブとジョブの詳細のリストが表示されます。 各ジョブが実行されているノード、そのジョブに割り当てられている GPU の数、ジョブの現在の実行時間、ジョブの進捗状況、およびそのジョブの GPU 利用率。1 つのチーム（「 TEAM 」）から送信された実行中のジョブは 3 つしかないため、クラスタの使用率が低いことに注意してください（ GPU 使用率は 23% ）。

次のセクションでは、「プロジェクト」タブで複数のチームを作成し、各チームに GPU を割り当てることで、クラスタあたりのユーザ数が多いときにクラスタの使用率を最大限に高め、リソースを管理する方法を説明します。テストシナリオは、トレーニング、推論、対話型のワークロードでメモリリソースと GPU リソースが共有されているエンタープライズ環境をシミュレートしたものです。

データサイエンスチームのプロジェクトを作成し、 GPU を割り当てる

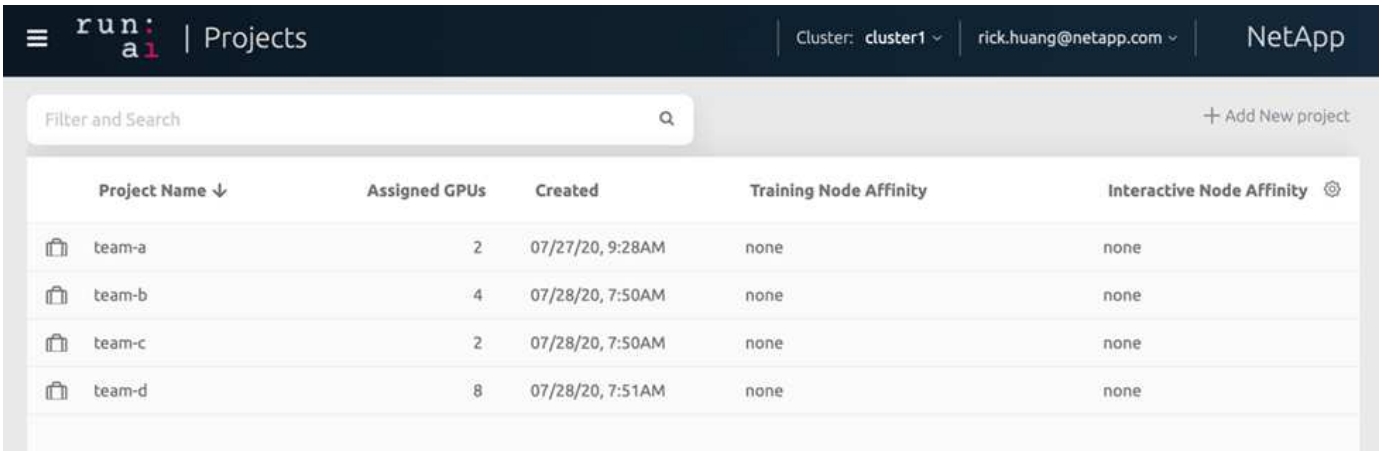
研究者は、 AI CLI や Kubeflow などのプロセスを使ってワークロードを送信できます。リソースの割り当てを合理化して優先順位を設定するために、 Run ： AI では「プロジェクト」という概念が導入されています。プロジェクトは、プロジェクト名を GPU 割り当てと優先設定に関連付けるクォータエンティティです。複数のデータサイエンスチームを管理するシンプルで便利な方法です。

ワークロードを送信する研究者は、プロジェクトをワークロード要求に関連付ける必要があります。Run ： AI スケジューラは、リクエストを現在の割り当てとプロジェクトと比較して、ワークロードにリソースを割り当てることができるかどうか、またはワークロードを保留中の状態にするかどうかを判断します。

システム管理者は、実行： AI プロジェクトタブで次のパラメータを設定できます。

- * モデルプロジェクト。 * ユーザーごとにプロジェクトを設定し、ユーザーのチームごとにプロジェクトを設定し、実際の組織プロジェクトごとにプロジェクトを設定します。
- * プロジェクトの割り当て量。 * 各プロジェクトは、このプロジェクトに同時に割り当てることができる GPU の割り当て量に関連付けられています。これは、このプロジェクトを使用している研究者が、クラスタ内のどの状態であっても、この数の GPU を取得できることが保証されるという意味で、保証されたクォータです。原則として、プロジェクトの割り当ての合計は、クラスタ内の GPU の数と等しくなければなりません。さらに、このプロジェクトのユーザは、過剰割り当てを受け取ることができます。GPU が使用されていない限り、このプロジェクトを使用する研究者は GPU を増やすことができます。に、クォータ超過テストのシナリオと公平性に関する考慮事項を示します "オーバークォータの GPU 割り当てによる高いクラスタ利用率の達成"、 "基本的な資源配分フェアネス"および "オーバークォータフェアネス"。
- 新しいプロジェクトを作成し、既存のプロジェクトを更新して、既存のプロジェクトを削除します。
- * 特定のノードグループで実行するジョブを制限 * 。特定のプロジェクトを割り当てて、特定のノードでのみ実行することができます。これは、プロジェクトチームが十分なメモリを備えた特殊なハードウェアを必要とする場合などに便利です。また、プロジェクトチームは、特別な予算で取得された特定のハードウェアの所有者になる場合もあります。また、パフォーマンスが低いハードウェアで作業したり、長いトレーニングや無人ワークロードを高速ノードに誘導したりするために、ビルドや対話型のワークロードを処理する必要がある場合もあります。ノードをグループ化し、特定のプロジェクトにアフィニティを設定するコマンドについては、を参照してください "「 AI Documentation 」を実行します"。
- * 対話型ジョブの期間を制限します * 。研究者たちは、対話型の仕事を閉じることを忘れがちだ。これにより、リソースの無駄が発生する可能性があります。一部の組織では、対話型ジョブの期間を制限し、自動的に終了することを希望しています。

次の図は、 4 つのチームが作成されたプロジェクトビューを示しています。各チームには、異なるワークロードに対応するために異なる数の GPU が割り当てられます。 GPU の総数は、 2 台の DGX-1 で構成されるクラスタ内の使用可能な GPU の総数と同じです。



実行中のジョブの送信： **AI CLI**

このセクションでは、 Kubernetes ジョブの実行に使用できる基本的な Run ： AI コマンドについて詳しく説明します。ワークロードの種類に応じて 3 つの要素に分割されます。AI / ML / DL のワークロードは、次の 2 つの汎用タイプに分けることができます。

- * 無人トレーニングセッション * 。このようなタイプのワークロードを扱うことで、データサイエンティストは自己実行ワークロードを準備し、実行のためにワークロードを送信します。実行中に、お客様は結果を確認できます。このタイプのワークロードは、多くの場合、本番環境で使用されます。また、モデル開発が段階で行われるため、手動操作は必要ありません。
- * インタラクティブビルドセッション * 。このようなワークロードの場合、データサイエンティストは Bash 、 Jupyter Notebook 、リモート PyCharm などの IDE を使用した対話型セッションを開始し、 GPU リソースに直接アクセスします。次に、対話型のワークロードを実行してポートを接続し、コンテナユーザに内部ポートを表示するシナリオを紹介します。

無人トレーニングのワークロード

プロジェクトをセットアップして GPU を割り当てたら、コマンドラインで次のコマンドを使用して Kubernetes のワークロードを実行できます。

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

このコマンドは、単一の GPU を割り当てたチーム A の無人トレーニングジョブを開始します。このジョブは、サンプルの Docker イメージである「 gcr.io/run-ai-demo/QuickStart 」に基づいています。私たちはジョブ「 hyper1 」と名付けました。その後、次のコマンドを実行して、ジョブの進捗状況を監視します。

```
$ runai list
```

次の図に 'runai list' コマンドの結果を示します一般的なステータスは次のとおりです。

- 「 ContainerCreating 」を参照してください。Docker コンテナをクラウドリポジトリからダウンロードしています。
- 「保留中」。ジョブはスケジュールされるのを待っています。

- 「ランニング」。ジョブが実行中です。

```
~> 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
```

ジョブのステータスをさらに表示するには、次のコマンドを実行します。

```
$ runai get hyper1
```

ジョブのログを表示するには、「 runai logs <job-name> 」コマンドを実行します。

```
$ runai logs hyper1
```

この例では、現在のトレーニングエポック、 ETA 、損失関数値、精度、および各ステップの経過時間を含む、実行中の DL セッションのログを確認する必要があります。

クラスタのステータスは、 Run ： AI UI で確認できます から "https://app.run.ai/"。[ダッシュボード]>[概要] で、 GPU 利用率を監視できます。

このワークロードを停止するには、次のコマンドを実行します。

```
$ runai delte hyper1
```

このコマンドを実行すると、トレーニングワークロードが停止します。このアクションを確認するには 'runai list' をもう一度実行します詳細については、を参照してください "無人トレーニングワークロードの開始"。

ワークロードを対話型に構築

プロジェクトをセットアップして GPU を割り当てたら、コマンドラインで次のコマンドを使用して対話型ビルドワークロードを実行できます。

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

このジョブは、サンプルの Docker イメージ Python に基づいています。ジョブ build1 という名前を付けました。

> 「 --interactive 」フラグは、ジョブが開始または終了していないことを意味します研究者の責任で業務を終了してください。管理者は、対話型ジョブがシステムによって終了されるまでの時間制限を定義できます。

--g 1' フラグは ' このジョブに 1 つの GPU を割り当てます与えられたコマンドと引数は、「 --command sleep — args インフィニティ」です。コマンドを指定するか、コンテナを開始してすぐに終了する必要があります。

次のコマンドは、で説明したコマンドと同様に機能します [無人トレーニングのワークロード]：

- runai list` ：名前、ステータス、経過時間、ノード、イメージ、 プロジェクト、ユーザ、 GPU によるジョブの処理
- runai get build1`: ジョブ build1 の追加ステータスを表示します。
- runai delete build1`: 対話型のワークロード build1 を停止しますコンテナに bash シェルを取得するには ' 次のコマンドを実行します

```
$ runai bash build1
```

これにより、コンピュータにシェルが直接挿入されます。データサイエンティストは、コンテナ内でモデルを開発または微調整できます。

クラスタのステータスは、 Run ： AI UI で確認できます から "https://app.run.ai"。詳細については、を参照してください "対話型ビルドワークロードの開始と使用"。

ポートが接続された対話型のワークロード

Run ： AI CLI でコンテナを開始する際に、対話型ビルドワークロードを拡張することで、コンテナユーザに内部ポートを公開できます。これは、クラウド環境、 Jupyter Notebook の操作、その他のマイクロサービスへの接続に役立ちます。 "入力" Kubernetes クラスタの外部から Kubernetes サービスへのアクセスを許可します。アクセスを設定するには、どのインバウンド接続がどのサービスに到達するかを定義する一連のルールを作成します。

クラスタ内のサービスへの外部アクセスを適切に管理するには、クラスタ管理者がインストールすることを推奨します "入力" ロードバランサを設定します。

入力をサービスタイプとして使用するには、次のコマンドを実行して、ワークロード送信時にメソッドタイプとポートを設定します。

```
$ 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
```

コンテナが正常に起動したら、「 runai list 」を実行して、 Jupyter Notebook にアクセスする「サービス URL (S) 」を確認します。URL は、入力エンドポイント、ジョブ名、およびポートで構成されます。たとえば、を参照してください https://10.255.174.13/test-ingress-8888。

詳細については、を参照してください "ポートが接続されている対話型のビルドワークロードを起動する"。

高いクラスタ利用率の達成

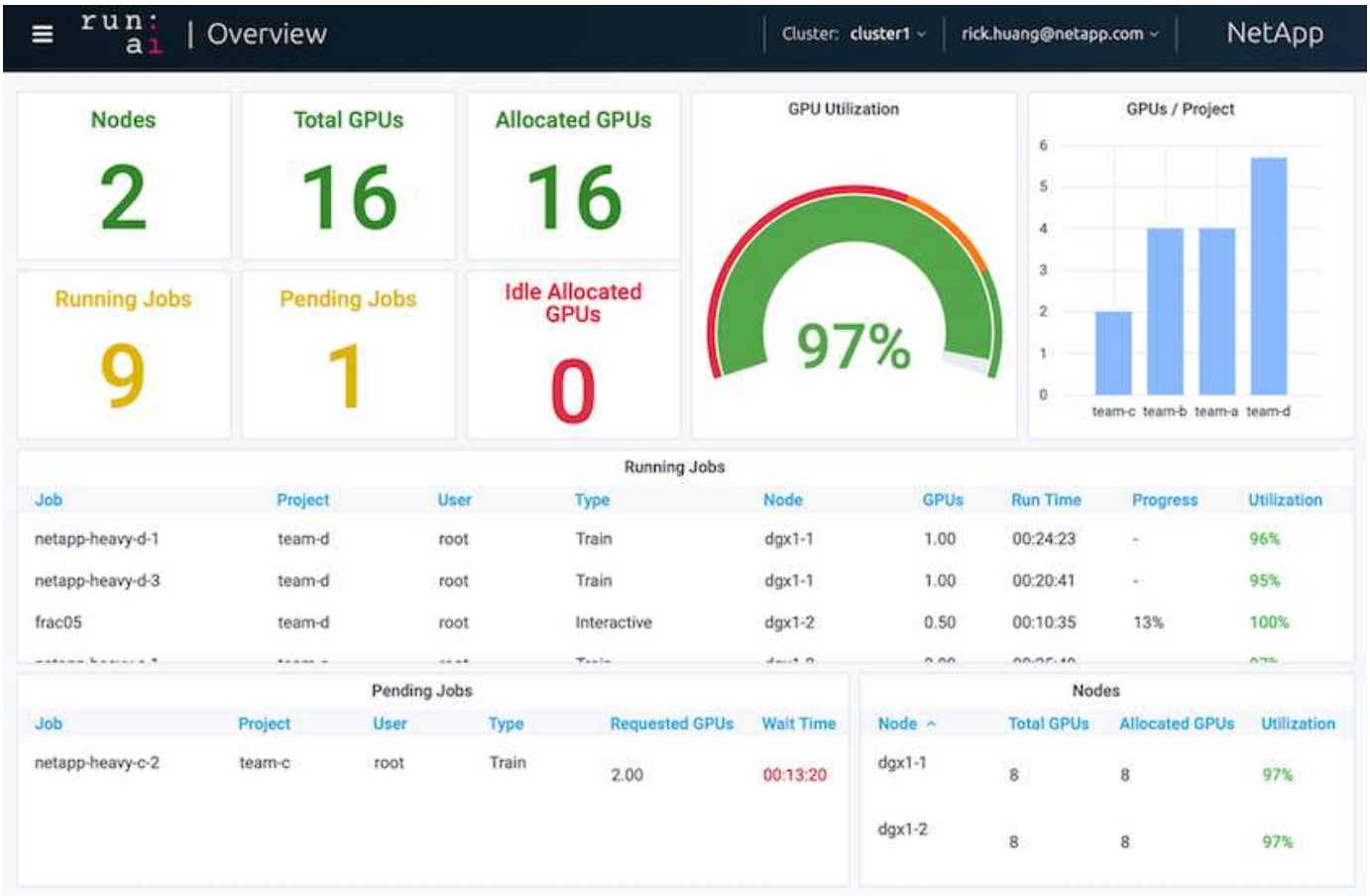
このセクションでは、 4 つのデータサイエンスチームがそれぞれ独自のワークロードを送信して実行： AI オーケストレーション解決策を実証する現実的なシナリオをエミュレートしています。このシナリオでは、 GPU リソースの優先順位付けとバランシングを維持しながら、クラスタの利用率を高めることができますまず、で説明した ResNet-50 ベンチマークを使用します セクション "ResNet-50 と ImageNet データセットベンチマ

ークの概要"：

```
$ 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
```

ResNet-50 ベンチマークを実行しました（を参照） "NVA-1121."。パブリック Docker リポジトリに存在しないコンテナには、フラグ「 --local-image 」を使用しました。ディレクトリ「 /mnt/' 」と「 /tmp' 」をホスト DGX-1 ノード上の「 /mnt/' 」と「 /tmp' 」にそれぞれコンテナにマウントしました。データセットは、ディレクトリを指す「 dataset_dir 」引数を持つ NetApp AFFA800 にあります。どちらの場合も '--num_devices =1' と '-g 1' は ' このジョブに 1 つの GPU を割り当てていることを意味します前者は「 run.py 」スクリプトの引数で、後者は「 runai submit 」コマンドのフラグです。

次の図は、 97% の GPU 利用率を備え、 16 個の使用可能な GPU が割り当てられたシステム概要ダッシュボードを示しています。GPU / プロジェクトの棒グラフでは、各チームに割り当てられている GPU の数を簡単に確認できます。[実行中のジョブ] ウィンドウ枠には、現在実行中のジョブ名、プロジェクト、ユーザー、タイプ、ノード、 GPU の消費、実行時間、進捗状況、利用率の詳細。キューに登録されているワークロードとその待機時間のリストが「保留中のジョブ」に表示されます。さらに、 Nodes ボックスは、クラスタ内の個々の DGX-1 ノードの GPU 番号と利用率を表示します。



パフォーマンスの低いワークロードやインタラクティブなワークロードに適した、フラクショナルな **GPU** 割り当て

開発、ハイパーパラメータチューニング、デバッグのどの段階であっても、研究者や開発者が自分のモデルに取り組んでいる場合、このようなワークロードに必要なコンピューティングリソースは通常少なくなります。したがって、フラクショナル GPU とメモリをプロビジョニングして、同じ GPU を他のワークロードに同時に割り当てることがより効率的です。実行： AI のオーケストレーション解決策は、 Kubernetes 上のコンテナ化されたワークロード向けのフラクショナル GPU 共有システムを提供します。CUDA プログラムを実行するワークロードをサポートするシステムで、推論やモデル構築などの軽量な AI タスクに特に適しています。フラクショナル GPU システムを使用すると、データサイエンスチームや AI エンジニアリングチームは、 1 つの GPU で複数のワークロードを同時に実行できます。これにより、コンピュータビジョン、音声認識、自然言語処理など、より多くのワークロードを同じハードウェア上で実行できるようになり、コストが削減されます。

実行： AI のフラクショナル GPU システムは、仮想化された論理 GPU を独自のメモリとコンピューティングスペースで効率的に作成します。このスペースは、コンテナが自己完結型のプロセッサであるかのように使用およびアクセスできます。これにより、複数のワークロードを同じ GPU 上で並行して実行でき、互いに影響することはありません。解決策は透過的でシンプル、かつ移植可能であり、コンテナ自体に変更を加える必要はありません。

一般的な usecase では、同じ GPU 上で 2 〜 8 個のジョブが実行されていることがわかります。つまり、同じハードウェアで作業を 8 倍行うことができます。

次の図のプロジェクト「 team -d 」に属するジョブ「分割 05 」については、割り当てられた GPU の数が 0.50 であることがわかります。さらに 'nvidia-smi コマンドによって確認されますこのコマンドは ' コンテナに使用できる GPU メモリが 'DGX-1 ノードの V100 GPU あたりの 32GB の半分である 16,255MB であることを示します

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

オーバークォータの **GPU** 割り当てによる高いクラスタ利用率の達成

を参照してください "基本的な資源配分フェアネス"および "オーバークォータフェアネス"また、高度なテストシナリオを考案し、複雑なワークロード管理、自動プリエンプティブスケジューリング、オーバークォータ GPU プロビジョニングを実現する Run ： AI オーケストレーション機能をデモしました。これにより、 ONTAP AI 環境でクラスタリソースの利用率を高め、エンタープライズレベルのデータサイエンスチームの生産性を最適化できました。

これらの 3 つのセクションで、次のプロジェクトとクォータを設定します。

プロジェクト	クォータ
チーム A	4.
チーム - b	2.
チーム -c	2.
チーム -d	8.

また、この 3 つのセクションでは、次のコンテナを使用します。

- Jupyter Notebook ： "jupyter/base-notebook
- 「 GCR.IO/run-ai-demo/QuickStart 」を実行します

このテストシナリオでは、次の目標を設定します。

- リソースのプロビジョニングの簡易性と、リソースの使用方法を説明します ユーザから抽象化されます
- ユーザが GPU のフラクションを簡単にプロビジョニングする方法を説明します GPU の数を整数で指定します
- チームを編成してコンピューティングのボトルネックを解消する方法を説明します リソースクォータがある場合は、ユーザがそのリソースクォータを確認します クラスタ内に GPU が搭載されている
- ネットアップコンテナなどの大量の計算処理を行うジョブを実行する際に、 NetApp 解決策を使用してデータパイプラインのボトルネックを解消する方法を説明する
- を使用して複数のタイプのコンテナを実行する方法を説明します システム
  - Jupyter ノートブック
  - 実行： AI コンテナ
- クラスタがフルの状態のときに高い利用率を表示します

テスト中に実行される実際のコマンドシーケンスの詳細については、を参照してください "セクション 4.8 のテストの詳細"。

13 個のワークロードすべてが送信されると、次の図に示すように、コンテナ名と割り当てられている GPU のリストが表示されます。7 つのトレーニングと 6 つの対話型ジョブが用意されており、 4 つのデータサイエンスチームをシミュレーションして、それぞれに独自のモデルを実行しているか開発中であるかを確認していますインタラクティブなジョブの場合、個々の開発者は Jupyter Notebook を使用してコードの書き込みやデバッグを行っています。そのため、クラスタリソースをあまり使用せずに GPU フラクションをプロビジョニングすることをお勧めします。

```
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
```

このテストシナリオの結果は次のようになります。

- クラスタがいっぱいである必要があります。 16 基の GPU が使用されています。
- クラスタの利用率が高い。
- フラクショナル割り当てにより、 GPU よりも多くの実験が行われています。
- 「 team -d 」では、すべてのクォータが使用されているわけではありません。したがって、「 team -b 」と「 team -c 」では、実験に追加の GPU を使用することができるため、イノベーションにかかる時間が短縮されます。

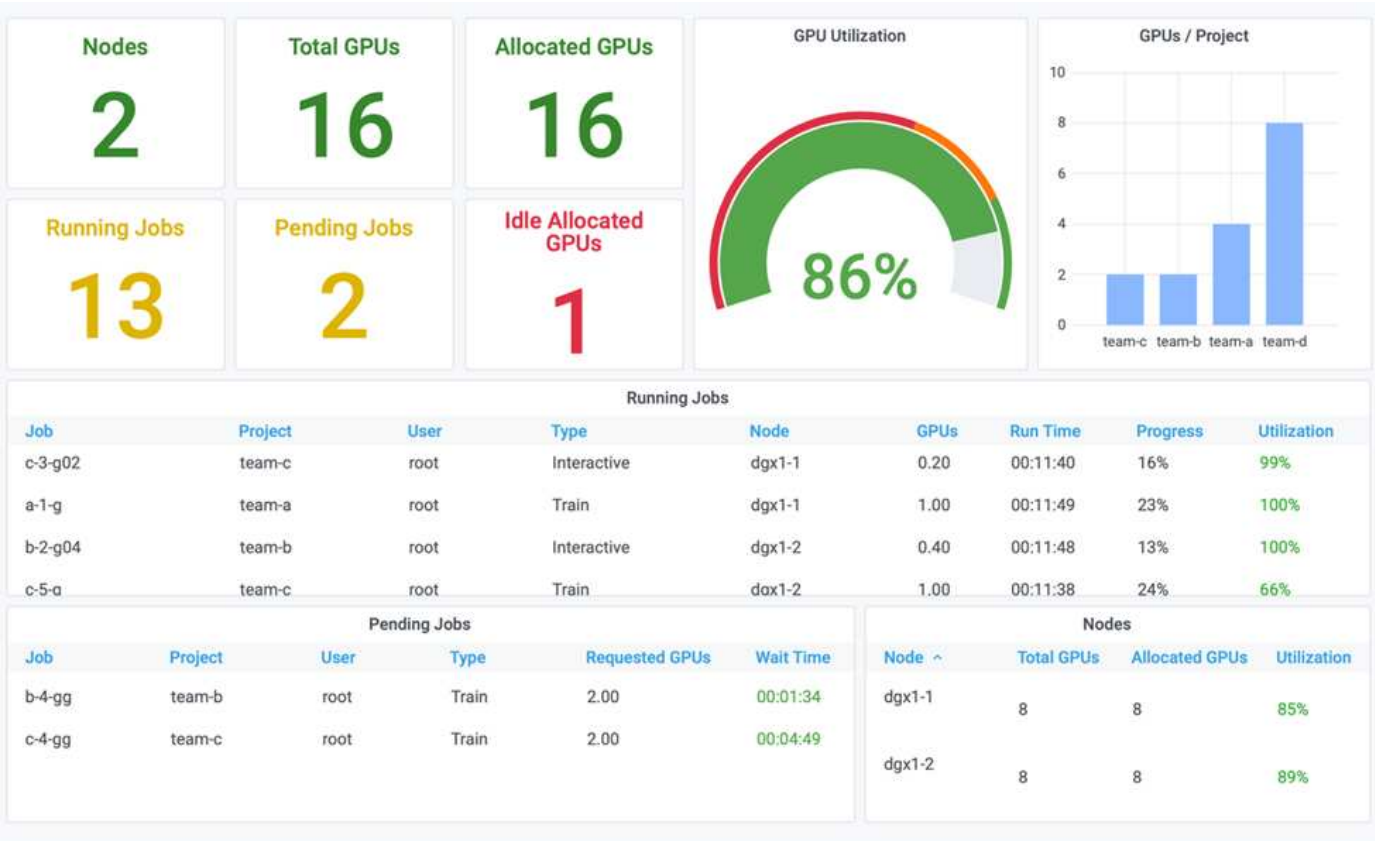
基本的な資源配分フェアネス

このセクションでは 'team -d がより多くの GPU を要求した場合 (それらは割り当ての

下にあります) ' システムは 'team -b' および 'team -c のワークロードを一時停止し ' 公正な共有方法で保留状態に移行することを示しています

ジョブの送信、使用するコンテナイメージ、実行するコマンドシーケンスなどの詳細については、を参照してください "セクション 4.9 のテストの詳細"。

次の図は、自動ロードバランシングとプリエンプティブスケジューリングにより、クラスタ使用率、チームごとに割り当てられた GPU 、保留中のジョブを示しています。すべてのチームワークロードが要求した GPU の総数が、クラスタ内で使用可能な合計 GPU 数を超えると、実行： AI の内部公正性アルゴリズムによって、「 team -b 」と「 team -c 」のそれぞれに 1 つのジョブが一時停止されます。これは、それらがプロジェクトの割り当て量を満たしているためです。これにより、クラスタ全体の利用率が向上しますが、データサイエンスチームは、管理者が設定したリソースの制約に基づいて引き続き作業できます。



このテストシナリオの結果は、次のことを示しています。

- * 自動ロードバランシング。 * システムは、各チームが割り当てを使用するように GPU の割り当て量を自動的に調整します。一時停止されていたワークロードは、そのクォータを超えていたチームに属しています。
- * 公正な共有の一時停止。 * システムは、ノルマを達成したチームのワークロードを停止してから、もう一方のチームのワークロードを停止します。実行： AI には内部的な公正性アルゴリズムがあります。

オーバークォータフェアネス

このセクションでは、複数のチームがワークロードを送信し、クォータを超過するシナリオを拡張します。この方法では、 Run ： AI の公正性アルゴリズムが、事前設定されたクォータの比率に従ってクラスタリソースを割り当てる方法を説明します。

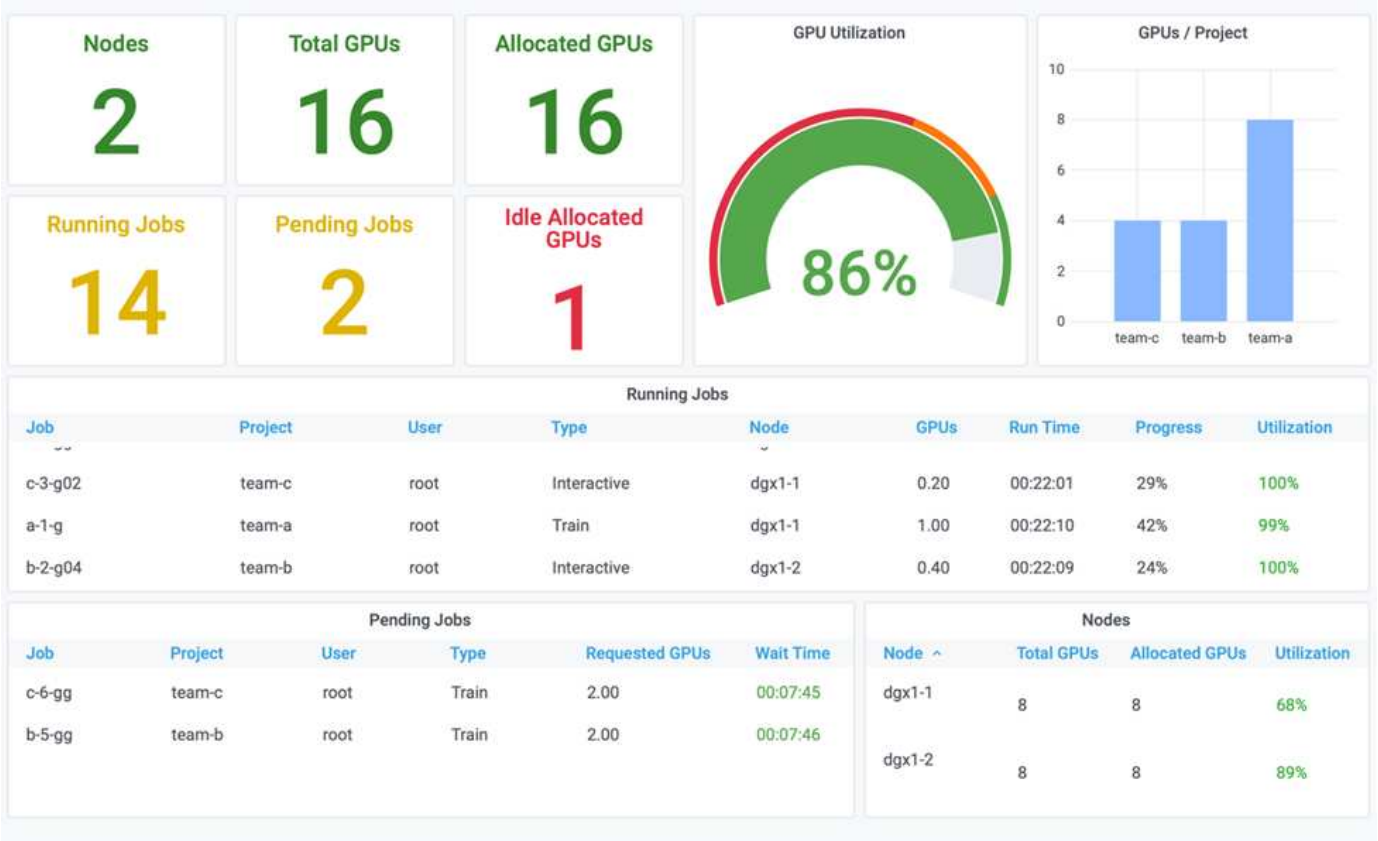
このテストシナリオの目標：

- 複数のチームがクォータを介して GPU を要求しているときのキューイングメカニズムを示します。
- システムが、クォータの比率に従って、クォータを超過した複数のチーム間にクラスタの適正な共有を分散し、クォータが大きいチームがスペア容量の大部分を占めるようにする方法を示します。

の末尾 "基本的な資源配分フェアネス"では、キューに入れられるワークロードは 2 つあります。 1 つは「 team -b 」用、もう 1 つは「 team -c 」用です。このセクションでは、追加のワークロードをキューに登録します。

ジョブの送信、使用されるコンテナイメージ、実行されるコマンドシーケンスなどの詳細については、を参照してください "セクション 4.10 のテストの詳細"。

すべてのジョブがセクションに従って送信される場合 "セクション 4.10 のテストの詳細"システム・ダッシュボードには 'team a'team -b'および 'team -c` のすべての GPU の数が ' 事前設定されたクォータよりも多くなっていることが示されています「 team -a 」は、初期設定のソフトクォータ（ 4 ）よりも 4 基の GPU を占有し、「 team -b 」と「 team -c 」はソフトクォータ（ 2 つ）よりも 2 つの GPU を占有します。割り当てられたクォータ超過 GPU の比率は、事前設定されたクォータの比率と同じです。これは、システムが優先順位の基準として事前設定されたクォータを使用し、複数のチームがクォータを超えて GPU を追加するように要求した場合に応じてプロビジョニングされるためです。このような自動ロードバランシングは、企業のデータサイエンスチームが AI モデルの開発と運用に積極的に関与している場合に、公平性と優先順位付けを提供します。



このテストシナリオの結果は次のようになります。

- 他のチームのワークロードのキュー解除が開始されます。
- キュー解除の順序は ' 公正性アルゴリズムに従って決定されますたとえば 'team -b と 'team -c は ' 同等のクォータを持つため ' 同じ量のオーバークォータ GPU を取得します また 'team -a は 'team -b と 'team -c

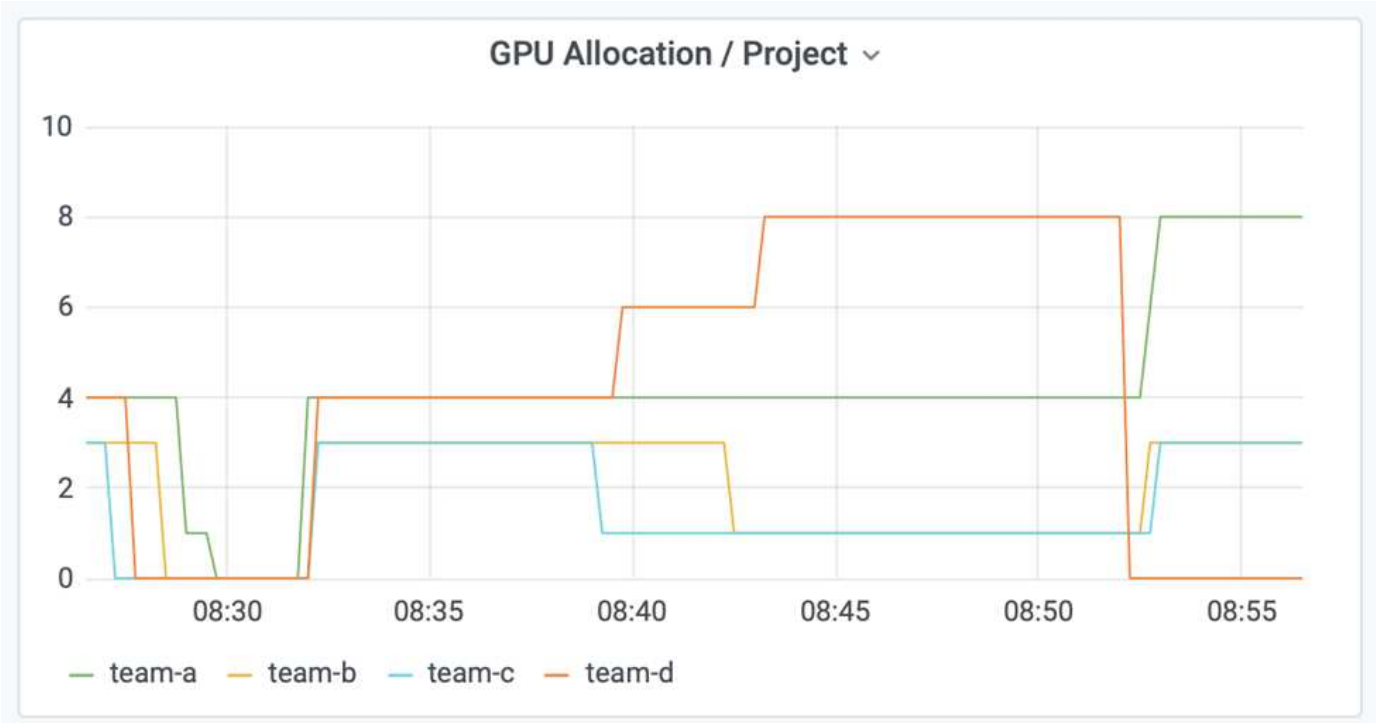
のクォータの 2 倍のクォータを備えているため 'GPU の量が 2 倍になります

- すべての割り当てが自動的に行われます。

したがって、システムは次の状態で安定します。

プロジェクト	**GPU** が割り当てられました	コメント（ **Comment** ）
チーム A	8/4	クォータを介した 4 基の GPU空のキューです。
チーム - b	4 月 2 日	クォータを介した 2 つの GPU 。1 つのワークロードがキューに登録
チーム -c	4 月 2 日	クォータを介した 2 つの GPU 。1 つのワークロードがキューに登録
チーム -d	0/8	GPU をまったく使用しないので、キューに登録されているワークロードはありません

次の図は、プロジェクトごとの GPU 割り当てを示しています Run ：セクションの AI Analytics ダッシュボードに表示される時間 "オーバークォータの GPU 割り当てによる高いクラスタ利用率の達成"、 "基本的な資源配分フェアネス"および "オーバークォータフェアネス"。図の各行は、特定のデータサイエンスチーム用にプロビジョニングされた GPU の数を常に表しています。システムは、送信されたワークロードに応じて GPU を動的に割り当てることがわかります。これにより、クラスタ内に使用可能な GPU がある場合はクォータを超過し、公平性に従ってジョブをプリエンプトしてから、 4 つのチームすべてが最終的に安定した状態に到達することができます。



Trident でプロビジョニングされた永続的ボリュームにデータを保存する

NetApp Trident は、コンテナ化されたアプリケーションが求める高度な永続性のニーズに対応できるように設計された、完全にサポートされているオープンソースプロジェク

トです。Trident でプロビジョニングされた Kubernetes PersistentVolume （ PV ）に対してデータの読み取りと書き込みを行うことができ、データ階層化、暗号化、 NetApp Snapshot テクノロジ、コンプライアンス、 NetApp ONTAP データ管理ソフトウェアが提供する高パフォーマンスといったメリットも活用できます。

既存の名前空間での **PVC** の再利用

大規模な AI プロジェクトでは、異なるコンテナで同じ Kubernetes PV に対してデータの読み取りや書き込みを行う方が効率的な場合があります。Kubernetes Persistent Volume Claim （ PVC ；永続ボリューム要求）を再利用するには、ユーザが PVC を作成しておく必要があります。を参照してください "NetApp Trident のドキュメント" PVC 作成の詳細については、を参照してください。次に、既存の PVC を再利用する例を示します。

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

次のコマンドを実行して ' プロジェクト 'team -a' のジョブ pvc-test' のステータスを表示します

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

'team -a' ジョブ 'pvctest' にマウントされた PV /tmp/pvc1mount が表示されますこのようにして、複数のコンテナが同じボリュームから読み取ることができるため、開発中または本番環境で競合する複数のモデルが存在する場合に便利です。データサイエンティストは、モデルのアンサンブルを構築し、大多数の投票またはその他の技術によって予測結果を組み合わせることができます。

次のコマンドを使用してコンテナシェルにアクセスします。

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

その後、マウントされたボリュームを確認し、コンテナ内のデータにアクセスできます。

PVC の再利用というこの機能は、 NetApp FlexVol ボリュームと NetApp ONTAP FlexGroup ボリュームと連携します。そのため、データエンジニアは、より柔軟で堅牢なデータ管理オプションを利用して、ネットアップのデータファブリックを活用できます。

まとめ

ネットアップと Run ：このテクニカルレポートでは、 AI と、 NetApp ONTAP AI 解決策の独自機能と、 Run ： AI プラットフォームを組み合わせることで、 AI ワークロードのオーケストレーションを簡易化できることを紹介しています。前の手順では、ディープラーニングのためのデータパイプラインとワークロードオーケストレーションのプロセスを合理化するリファレンスアーキテクチャを示します。これらのソリューションの導入を検討しているお客様には、ネットアップと Run ： AI の活用で詳細を確認することをお勧めします。

セクション 4.8 のテストの詳細

ここでは、のテストの詳細について説明します "オーバークォータの GPU 割り当てによる高いクラスタ利用率の達成"。

次の順序でジョブを送信します。

プロジェクト	イメージ（ **Image** ）	**GPU** の数	合計	コメント（ **Comment** ）
チーム A	Jupyter	1.	1/4	–
チーム A	ネットアップ	1.	2/4	–
チーム A	実行： AI	2.	4 月 4 日	すべてのクォータを使用しています
チーム - b	実行： AI	0.6	0.6/2	フラクショナル GPU
チーム - b	実行： AI	0.4	1/2	フラクショナル GPU
チーム - b	ネットアップ	1.	2/2.	–
チーム - b	ネットアップ	2.	4 月 2 日	クォータに 2 つ
チーム -c	実行： AI	0.5	0.5/2	フラクショナル GPU
チーム -c	実行： AI	0.3	0.8/2.	フラクショナル GPU
チーム -c	実行： AI	0.2	1/2	フラクショナル GPU
チーム -c	ネットアップ	2.	3/2	クォータに 1 つ
チーム -c	ネットアップ	1.	4 月 2 日	クォータに 2 つ
チーム -d	ネットアップ	4.	4/8	クォータの半分を使用します

コマンドの構造：

```
$ runai submit <job-name> -p <project-name> -g <#GPUs> -i <image-name>
```

テストで使用する実際のコマンドシーケンス：

```
$ runai submit a-1-1-jupyter -i jupyter/base-notebook -g 1 \
  --interactive --service-type=ingress --port 8888 \
  --args="--NotebookApp.base_url=team-a-test-ingress" --command=start
-notebook.sh -p team-a
$ runai submit a-1-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-a
$ runai submit a-2-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a
$ runai submit b-1-g06 -i gcr.io/run-ai-demo/quickstart -g 0.6
--interactive -p team-b
$ runai submit b-2-g04 -i gcr.io/run-ai-demo/quickstart -g 0.4
--interactive -p team-b
$ runai submit b-3-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-b
$ runai submit b-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-b
$ runai submit c-1-g05 -i gcr.io/run-ai-demo/quickstart -g 0.5
--interactive -p team-c
$ runai submit c-2-g03 -i gcr.io/run-ai-demo/quickstart -g 0.3
--interactive -p team-c
$ runai submit c-3-g02 -i gcr.io/run-ai-demo/quickstart -g 0.2
--interactive -p team-c
$ runai submit c-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-c
$ runai submit c-5-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-c
$ runai submit d-1-gggg -i gcr.io/run-ai-demo/quickstart -g 4 -p team-d
```

この時点で、次の状態になります。

プロジェクト	**GPU** が割り当てられました	ワークロードキュー
チーム A	4/4 （ソフトクォータ / 実際の割り当て）	なし
チーム - b	4 月 2 日	なし
チーム -c	4 月 2 日	なし
チーム -d	4/8	なし

を参照してください "「 Achieving High Cluster Utilization With over-uota 」 GPU 割り当て" 進行中のテストシナリオについてのディスカッションにご参加ください。

セクション **4.9** のテストの詳細

ここでは、のテストについて詳しく説明します "基本的な資源配分フェアネス"。

次の順序でジョブを送信します。

プロジェクト	**GPU** の数	合計	コメント（**Comment**）
チーム -d	2.	6 月 8 日	team -b/c ワークロードが一時停止し、「 pending 」に移動します。
チーム -d	2.	8 月 8 日	他のチーム (b/c) のワークロードは一時停止し、「保留中」に移動します。

実行される次のコマンドシーケンスを参照してください。

```
$ runai submit d-2-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-d$
runai submit d-3-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-d
```

この時点で、次の状態になります。

プロジェクト	**GPU** が割り当てられました	ワークロードキュー
チーム A	4 月 4 日	なし
チーム - b	2/2.	なし
チーム -c	2/2.	なし
チーム -d	8 月 8 日	なし

を参照してください "基本的な資源配分フェアネス" を参照してください。

セクション **4.10** のテストの詳細

ここでは、のテストについて詳しく説明します "オーバークォータフェアネス"。

「 team -a 」、「 team -b 」、「 team -c 」の順にジョブを送信します。

プロジェクト	**GPU** の数	合計	コメント（**Comment**）
チーム A	2.	4 月 4 日	1 個のワークロードをキューに登録
チーム A	2.	4 月 4 日	2 個のワークロードがキューに登録
チーム - b	2.	2/2.	2 個のワークロードがキューに登録
チーム -c	2.	2/2.	2 個のワークロードがキューに登録

実行される次のコマンドシーケンスを参照してください。

```
$ runai submit a-3-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a$
runai submit a-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a$ runai
submit b-5-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-b$ runai
submit c-6-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-c
```

この時点で、次の状態になります。

プロジェクト	**GPU** が割り当てられました	ワークロードキュー
チーム A	4 月 4 日	2 つのワークロードがそれぞれ 2 つの GPU を必要とします
チーム - b	2/2.	2 つのワークロードがそれぞれ 2 つの GPU を必要とします
チーム -c	2/2.	2 つのワークロードがそれぞれ 2 つの GPU を必要とします
チーム -d	8 月 8 日	なし

次に 'team -d のすべてのワークロードを削除します

```
$ runai delete -p team-d d-1-gggg d-2-gg d-3-gg
```

を参照してください "オーバークォータフェアネス"を参照してください。

追加情報の検索場所

このドキュメントに記載されている情報の詳細については、次のリソースを参照してください。

- NVIDIA DGX システム
  - NVIDIA DGX-1 システムhttps://www.nvidia.com/en-us/data-center/dgx-1/[]
  - NVIDIA V100 Tensor コア GPUhttps://www.nvidia.com/en-us/data-center/tesla-v100/[]
  - NVIDIA NGChttps://www.nvidia.com/en-us/gpu-cloud/[]
- 実行： AI コンテナオーケストレーション解決策
  - 実行： AI 製品の概要https://docs.run.ai/home/components/[]
  - 実行： AI インストールドキュメントhttps://docs.run.ai/Administrator/Cluster-Setup/Installing-Run-AI-on-an-on-premise-Kubernetes-Cluster/[] https://docs.run.ai/Administrator/Researcher-Setup/Installing-the-Run-AI-Command-Line-Interface/
  - 実行時のジョブの送信： AI CLIhttps://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Launch-Unattended-Training-Workloads-/[] https://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Start-and-Use-Interactive-Build-Workloads-/
  - 実行時の GPU フラクションの割り当て： AI CLIhttps://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Using-GPU-Fractions/[]

- NetApp AI コントロールプレーン
  - テクニカルレポートをご参照くださいhttps://www.netapp.com/us/media/tr-4798.pdf[]
  - 簡単なデモhttps://youtu.be/gfr_sO27Rvo[]
  - GitHub リポジトリhttps://github.com/NetApp/kubeflow_jupyter_pipeline[]
- NetApp AFF システム
  - NetApp AFF A シリーズのデータシートhttps://www.netapp.com/us/media/ds-3582.pdf[]
  - ネットアップの All Flash FAS 向けフラッシュソリューションの利点https://www.netapp.com/us/media/ds-3733.pdf[]
  - ONTAP 9 情報ライブラリhttp://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286[]
  - NetApp ONTAP FlexGroup Volume テクニカルレポートhttps://www.netapp.com/us/media/tr-4557.pdf[]
- NetApp ONTAP AI
  - DGX-1 と Cisco Networking Design Guide による ONTAP AIhttps://www.netapp.com/us/media/nva-1121-design.pdf[]
  - DGX-1 と Cisco Networking Deployment Guide を使用した ONTAP AIhttps://www.netapp.com/us/media/nva-1121-deploy.pdf[]
  - DGX-1 と Mellanox のネットワーキング設計ガイドで構成される ONTAP AIhttp://www.netapp.com/us/media/nva-1138-design.pdf[]
  - DGX-2 を使用した ONTAP AI 設計ガイドhttps://www.netapp.com/us/media/nva-1135-design.pdf[]