Red Hat OpenShift AI Self-Managed 2.9

分散ワークロードの使用

分散ワークロードを使用して、より高速で効率的なデータ処理とトレーニングのモデル化

Last Updated: 2024-05-04

Red Hat OpenShift AI Self-Managed 2.9 分散ワークロードの使用

分散ワークロードを使用して、より高速で効率的なデータ処理とトレーニングのモデル化

法律上の通知

概要

分散ワークロードにより、データサイエンティストは複数のクラスターノードを並行して使用して、より高速かつ効率的なデータ処理とモデルトレーニングを行うことができます。CodeFlare フレームワークは、タスクのオーケストレーションと監視を簡素化し、高度な GPU サポートによる自動リソーススケーリングと最適なノード利用のためのシームレスな統合を提供します。

目次

はじめに

複雑な機械学習モデルのトレーニングや、データのより迅速な処理を行えるように、データサイエンティストは分散ワークロード機能を使用して、複数の OpenShift ワーカーノードでジョブを並行して実行できます。このアプローチにより、タスクの完了時間が大幅に短縮され、より大規模なデータセットとより複雑なモデルの使用が可能になります。

第1章 分散ワークロードの概要

分散ワークロード機能を使用すると、OpenShift クラスター内の複数のノード間でデータサイエンスワークロードの同時実行に必要なリソースをキューに入れ、スケールし、管理できます。通常、データサイエンスワークロードには、機械学習 (ML) や Python ワークロードなど、数種類の人工知能 (AI) ワークロードが含まれます。

分散ワークロードには、次の利点があります。

- 処理時間が短縮されるため、より速く反復し、より頻繁に実験できます。
- より大きなデータセットを使用できるため、より正確なモデルが得られます。
- 単一ノードではトレーニングできなかった複雑なモデルを使用できます。
- 分散ワークロードをいつでも送信でき、システムは必要なリソースが利用可能になったときに分散ワークロードをスケジュールできます。

分散ワークロードインフラストラクチャーには次のコンポーネントが含まれます。

CodeFlare Operator

デプロイされた Ray クラスターを保護し、それらの URL へのアクセスを許可します。

CodeFlare SDK

Python ベースの環境のリモート分散コンピュートジョブとインフラストラクチャーを定義し、制御します。

KubeRay

分散コンピューティングワークロードを実行するために OpenShift 上のリモート Ray クラスターを管理します

Kueue

クォータを管理し、分散ワークロードがそれらを消費する方法を管理し、クォータに関して分散ワークロードのキューイングを管理します。

分散ワークロードは、データサイエンスパイプライン、Jupyter ノートブック、または Microsoft Visual Studio Code ファイルから実行できます。

第2章 分散ワークロードの設定

データサイエンティストが OpenShift AI で使用する分散ワークロード機能を設定するには、必要な Kueue リソースを作成し、Red Hat OpenShift AI Operator で複数のコンポーネントを有効にし、必要に応じて CodeFlare Operator を設定する必要があります。

2.1. 分散ワークロードコンポーネントの設定

データサイエンティストが OpenShift AI で使用できるように分散ワークロード機能を設定するには、いくつかのコンポーネントを有効にする必要があります。

前提条件

- **cluster-admin** ロールで OpenShift Container Platform にログインしている。
- データサイエンスクラスターにアクセスできる。
- Red Hat OpenShift AI をインストールしている。
- 十分なリソースがある。OpenShift AI の インストールとデプロイ で説明されている最小 OpenShift AI リソース（非 接続環境での OpenShift AIのデプロイ を参照してください）のほかに、分散ワークロードインフラストラクチャーをデプロイするには 1.6 vCPU および 2 GiB メモリーが必要です。
- Ray クラスターイメージにアクセスできる。Ray クラスターの作成方法については、Ray クラスターのドキュメント を参照してください。

注記

相互トランスポート層セキュリティー(mTLS)は、OpenShift AI の CodeFlare コンポーネントでデフォルトで有効になっています。現在の OpenShift AI バージョンでは、Submit **Mode=K8sJobMode** は Ray ジョブ仕様ではサポートされていません。そのため、KubeRay Operator は送信者の Kubernetes ジョブを作成して Ray ジョブを送信できません。代わりに、ユーザーは、Submit **Mode=HTTPMode** のみを設定するように Ray ジョブ仕様を設定して、KubeRay Operator が RayCluster にリクエストを送信して Ray ジョブを作成できるようにする必要があります。

- 分散されたワークロードが使用するデータセットおよびモデルにアクセスできる。
- 分散されたワークロードの Python 依存関係にアクセスできる。
- ナレッジベースのソリューション記事 How to migrate from a separately installed CodeFlare Operator in your data science cluster の説明に従って、以前にインストールされた CodeFlare Operator のインスタンスがすべて削除されている。
- グラフィックスプロセッシングユニット (GPU) を使用する場合は、OpenShift AI で GPU サポートを有効にしている。OpenShift AI での GPU サポートの有効化 を参照してください。
- 自己署名証明書を使用する場合は、証明書の使用 で説明されているように、証明書を中央証明局 (CA) バンドルに追加しておきます (非接続環境の場合は、証明書の使用 を参照してください)。分散ワークロードでこれらの証明書を使用するために必要な追加設定はありません。一元的に設定された自己署名付き証明書は、次のマウントポイントのワークロード Pod で自動的に使用できるようになります。

- クラスター全体の CA バンドル:

```
/etc/pki/tls/certs/odh-trusted-ca-bundle.crt
/etc/ssl/certs/odh-trusted-ca-bundle.crt
```

- カスタム CA バンドル:

```
/etc/pki/tls/certs/odh-ca-bundle.crt
/etc/ssl/certs/odh-ca-bundle.crt
```

手順

1. OpenShift Container Platform コンソールで、**Operators → Installed Operators** をクリックします。
2. **Red Hat OpenShift AI** Operator を検索し、Operator 名をクリックして Operator details ページを開きます。
3. **Data Science Cluster** タブをクリックします。
4. デフォルトのインスタンス名（例： **default-dsc**）をクリックし、インスタンスの詳細ページを開きます。
5. **YAML** タブをクリックして、インスタンスの仕様を表示します。
6. 必要な分散ワークロードコンポーネントを有効にします。**spec:components** セクションで、必要なコンポーネントの **managementState** フィールドを正しく設定します。必要なコンポーネントの一覧は、次の表に示すように、分散ワークロードがパイプラインまたはノートブックから実行されるか、またはその両方から実行されるかによって異なります。

表2.1 分散ワークロードに必要なコンポーネント

コンポーネント	パイプラインのみ	ノートブックのみ	パイプラインおよびノートブック
codeflare	**Managed**	**Managed**	**Managed**
dashboard	**Managed**	**Managed**	**Managed**
datasciencepipelines	**Managed**	**Removed**	**Managed**
kueue	**Managed**	**Managed**	**Managed**
ray	**Managed**	**Managed**	**Managed**
workbenches	**Removed**	**Managed**	**Managed**

7. **Save** をクリックします。しばらくすると、**Managed** 状態のコンポーネントの準備が整います。

検証

以下のように、**codeflare-operator-manager**、**kuberay-operator**、および **kueue-controller-manager** Pod のステータスを確認します。

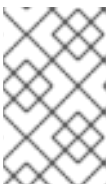
1. OpenShift Container Platform コンソールの **Project** リストから、**redhat-ods-applications** を選択します。
2. **Workloads → Deployments** をクリックします。
3. **codeflare-operator-manager**、**kuberay-operator**、および **kue-controller-manager** デプロイメントを検索します。それぞれのケースで以下のようにステータスを確認してください。
   a. デプロイメント名をクリックして、デプロイメントの詳細ページを開きます。
   b. **Pods** タブをクリックします。
   c. Pod のステータスを確認します。
   codeflare-operator-manager- <pod-id>、**kuberay- operator- <pod-id>**、**および kue-controller-manager- <pod-id** > Pod のステータスが **Running** の場合、Pod は使用可能な状態になります。
   d. 各 Pod の詳細を表示するには、Pod 名をクリックして Pod details ページを開き、**Logs** タブをクリックします。

2.2. 分散ワークロードのクォータ管理の設定

複数のデータサイエンスプロジェクト間でリソースを共有できるように、クラスター上で分散ワークロードのクォータを設定します。

前提条件

- OpenShift Container Platform クラスターのクラスター管理者権限を持っている。
- OpenShift コマンドラインインターフェイス (CLI) をダウンロードしてインストールしている。OpenShift CLI のインストール を参照してください。
- 分散ワークロードコンポーネント の設定 で説明されているように、必要な分散ワークロードコンポーネントを 有効にしている。
- 十分なリソースがある。ベースの OpenShift AI リソースに加えて、分散ワークロードインフラストラクチャーをデプロイするには 1.1 vCPU および 1.6 GB のメモリーが必要です。
- リソースはクラスター内で物理的に利用可能です。



注記

OpenShift AI は現在、クラスターごとに 1 つのクラスターキュー（つまり、同種のクラスター）のみをサポートし、空のリソースフレーバーのみをサポートしています。Kueue リソースの詳細は、Kueue ドキュメント を参照してください。

手順

1. ターミナルウィンドウで、クラスター管理者として OpenShift クラスターにまだログインしていない場合は、次の例に示すように OpenShift CLI にログインします。

```
$ oc login <openshift_cluster_url> -u <admin_username> -p <password>
```

2. 以下のように、空の Kueue リソースフレーバーを作成します。

   a. **default_flavor.yaml** という名前のファイルを作成し、以下の内容で入力します。

   空の Kueue リソースフレーバー

   ```
   apiVersion: kueue.x-k8s.io/v1beta1
   kind: ResourceFlavor
   metadata:
     name: default-flavor
   ```

   b. 設定を適用して **default-flavor** オブジェクトを作成します。

   ```
   $ oc apply -f default_flavor.yaml
   ```

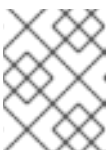
3. 以下のように、空の Kueue リソースフレーバーを管理するためのクラスターキューを作成します。

   a. **cluster_queue.yaml** という名前のファイルを作成し、以下の内容を設定します。

   クラスターキューの例

   ```
   apiVersion: kueue.x-k8s.io/v1beta1
   kind: ClusterQueue
   metadata:
     name: "cluster-queue"
   spec:
     namespaceSelector: {}  # match all.
     resourceGroups:
     - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
       flavors:
       - name: "default-flavor"
         resources:
         - name: "cpu"
           nominalQuota: 9
         - name: "memory"
           nominalQuota: 36Gi
         - name: "nvidia.com/gpu"
           nominalQuota: 5
   ```

   b. サンプルクォータ値(9 CPU、36 GiB メモリー、および 5 NVIDIA GPU)は、クラスターキューに適切な値に置き換えます。クラスターキューは、必要な合計リソースがこれらのクォータ制限内にある場合にのみ、分散ワークロードを開始します。

   注記

   本リリースの OpenShift AI では、分散ワークロードでサポートされる唯一のアクセラレーターは NVIDIA GPU です。

   c. 設定を適用して **cluster-queue** オブジェクトを作成します。

   ```
   $ oc apply -f cluster_queue.yaml
   ```

4. 以下のように、クラスターキューを参照するローカルキューを作成します。

a. **local_queue.yaml** という名前のファイルを作成し、以下の内容を設定します。

ローカルキューの例

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: LocalQueue
metadata:
  namespace: test
  name: local-queue-test
  annotations:
    kueue.x-k8s.io/default-queue: 'true'
spec:
  clusterQueue: cluster-queue
```

kue.x-k8s.io/default-queue: 'true' アノテーションは、このキューをデフォルトキューとして定義します。データサイエンスパイプライン、Jupyter ノートブック、または Microsoft Visual Studio Code ファイルの **ClusterConfiguration** セクションで **local_queue** 値が指定されていない場合、分散ワークロードはこのキューに送信されます。

b. **namespace** の値を更新して、Ray クラスターを作成する **ClusterConfiguration** セクションと同じ namespace を指定します。

c. オプション： **name** の値を更新します。

d. 設定を適用して local-queue オブジェクトを作成します。

```
$ oc apply -f local_queue.yaml
```

クラスターキューは、ローカルキューで分散ワークロードを実行するリソースを割り当てます。

検証

以下のように、プロジェクト内のローカルキューのステータスを確認します。

```
$ oc get -n <project-name> localqueues
```

関連情報

- Kueue ドキュメント

2.3. CODEFLARE OPERATOR の設定

OpenShift AI で分散ワークロードの CodeFlare Operator のデフォルト設定を変更する場合は、関連する設定マップを編集できます。

前提条件

- **cluster-admin** ロールで OpenShift Container Platform にログインしている。
- 分散ワークロードコンポーネント の設定 で説明されているように、必要な分散ワークロードコンポーネントを 有効にしている。

手順

1. OpenShift Container Platform コンソールで **Workloads → ConfigMaps** をクリックします。
2. **Project** リストから **redhat-ods-applications** を選択します。
3. **codeflare-operator-config** 設定マップを検索し、設定マップ名をクリックして **ConfigMap の詳細** ページを開きます。
4. **YAML** タブをクリックして設定マップの仕様を表示します。
5. **data:config.yaml:kuberay** セクションで、次のエントリーを編集できます。

ingressDomain

この設定オプションはデフォルトで null (**ingressDomain: ""**)です。Ingress コントローラーが OpenShift で実行されていない限り、このオプションを変更しないでください。OpenShift AI は、次の例に示すように、この値を使用してすべての Ray Cluster のダッシュボードとクライアントルートを生成します。

ダッシュボードとクライアントルートの例

```
ray-dashboard-<clustername>-<namespace>.<your.ingress.domain>
ray-client-<clustername>-<namespace>.<your.ingress.domain>
```

mTLSEnabled

この設定オプションはデフォルトで有効になっています(**mTLSEnabled: true**)。このオプションを有効にすると、Ray Cluster Pod は Ray Cluster ノード間に相互トランスポート層セキュリティー(mTLS)に使用される証明書を作成します。このオプションを有効にすると、Ray クライアントは、生成された証明書を **ca-secret-_<cluster_name>_ シークレット** からダウンロードし、mTLS 通信に必要な証明書を生成し、必要な Ray 環境変数を設定する場合を除き、Ray ヘッドノードに接続することができません。次に、ユーザーは Ray クライアントを再初期化して変更を適用する必要があります。CodeFlare SDK は、Ray クライアントの認証プロセスを簡素化する次の機能を提供します。

Ray クライアント認証コードの例

```
from codeflare_sdk import generate_cert

generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace)
generate_cert.export_env(cluster.config.name, cluster.config.namespace)

ray.init(cluster.cluster_uri())
```

rayDashboardOauthEnabled

この設定オプションは、デフォルトで有効になっています(**rayDashboardOAuthEnabled: true**)。このオプションを有効にすると、OpenShift AI は OpenShift OAuth プロキシーを Ray Cluster head ノードの前に配置します。その後、ユーザーはブラウザー経由で Ray Dashboard にアクセスする際に OpenShift クラスターのログイン認証情報を使用して認証する必要があります。ユーザーが別の方法で Ray ダッシュボードにアクセスする必要がある場合（例：Ray **JobSubmissionClient** クラスを使用）、以下の例のように認証ヘッダーを要求の一部として設定する必要があります。

認証ヘッダーの例

```
{Authorization: "Bearer <your-openshift-token>"}
```

6. Save をクリックして変更を **保存** します。

7. 変更を適用するには、Pod を削除します。

   a. **Workloads → Pods** をクリックします。

   b. **codeflare-operator-manager-<pod-id> Pod を検索** します。

   c. その Pod のオプションメニュー(⋮)をクリックし、**Delete Pod** をクリックします。変更を適用して Pod を再起動します。

検証

以下のように **codeflare-operator-manager** Pod のステータスを確認します。

1. OpenShift Container Platform コンソールで **Workloads → Deployments** をクリックします。

2. **codeflare-operator-manager** デプロイメントを検索し、デプロイメント名をクリックしてデプロイメントの詳細ページを開きます。

3. **Pods** タブをクリックします。codeflare-operator-manager-_<pod-id>_ Pod のステータスが **Running** になると、Pod は使用できる状態になります。Pod に関する詳細情報を表示するには、Pod 名をクリックして Pod の詳細ページを開き、Logs タブをクリックします。

第3章 分散ワークロードの実行

OpenShift AI では、ノートブックまたはパイプラインから分散ワークロードを実行できます。必要なソフトウェアすべてにアクセスできる場合は、切断された環境で分散ワークロードを実行することもできます。

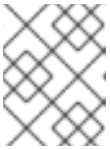
3.1. ノートブックからの分散データサイエンスワークロードの実行

ノートブックから分散データサイエンスワークロードを実行するには、まず Ray クラスターイメージへのリンクを提供する必要があります。

前提条件

- 分散ワークロードの設定 の説明に従って分散ワークロードを実行するように設定したデータサイエンスクラスターにアクセスできる。
- 分散ワークロードのクォータ管理の設定 の説明に従って、必要な Kueue リソースを 作成している。
- オプション： 分散ワークロードのクォータ管理の設定 で説明されているように、**LocalQueue** リソースを作成し、次のアノテーションを設定詳細に追加して、Ray クラスターの **デフォルト** のローカルキューを 定義している。

```
"kueue.x-k8s.io/default-queue": "true"
```



注記

デフォルトのローカルキューを作成しない場合は、各ノートブックにローカルキューを指定する必要があります。

- デフォルトのノートブックイメージの 1 つ (例: **Standard Data Science** ノートブック) を実行するワークベンチを含むデータサイエンスプロジェクトを作成している。デフォルトのノートブックの完全なリストは、データサイエンティスト向けのノートブックイメージ の表を参照してください。
- ノートブックサーバーを起動し、ノートブックエディターにログインしている。この手順の例では、JupyterLab 統合開発環境(IDE)を参照しています。

手順

1. CodeFlare SDK が提供するデモノートブックをダウンロードします。デモノートブックは、独自のノートブックで CodeFlare スタックを使用する方法のガイドラインを提供します。
   デモノートブックにアクセスするには、次のように **codeflare-sdk** リポジトリーのクローンを作成します。
   a. JupyterLab インターフェイスで、**Git > Clone a Repository**をクリックします。
   b. リポジトリーのクローン作成ダイアログで、**https://github.com/project-codeflare/codeflare-sdk.git** と入力し、**クローン** をクリックします。codeflare-sdk リポジトリーは左側のナビゲーションペインに一覧表示されます。
2. 以下の例のように分散ワークロードジョブを実行します。

a. JupyterLab インターフェイスで、左側のナビゲーションペインで **codeflare-sdk** をダブルクリックします。

b. **demo-notebooks** を ダブルクリックし、次に **guided-demos** をダブルクリックします。

c. 各サンプルデモノートブックを次のように更新します。

- まだ指定されていない場合は、**import** セクションを更新して **generate_cert** コンポーネントをインポートします。

更新されたインポートセクション

```
from codeflare_sdk import generate_cert
```

- default namespace の値をデータサイエンスプロジェクトの名前に置き換えます。
- サンプルコミュニティーイメージへのリンクは、Ray クラスターイメージへのリンクに置き換えます。
- Ray cluster creation セクションの後、以下の Ray クラスター認証コードが含まれていることを確認します。

レイのクラスター認証コード

```
generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace)
generate_cert.export_env(cluster.config.name, cluster.config.namespace)
```

注記

相互トランスポート層セキュリティー(mTLS)は、OpenShift AI の CodeFlare コンポーネントでデフォルトで有効になっています。ノートブック内で実行される Ray クライアントが、mTLS が有効になっているセキュアな Ray クラスターに接続できるようにするには、ベイのクラスター認証コードを含める必要があります。

- 分散ワークロードのクォータ管理の設定 の説明に従って、**kue.x-k8s.io/default-queue: 'true'** アノテーションを含めてデフォルトのローカルキューを設定 していない場合は、以下の例のように **ClusterConfiguration** セクションを更新して Ray クラスターのローカルキューを指定します。

ローカルキューの割り当て例

```
local_queue="your_local_queue_name"
```

d. ノートブックを実行します。

検証

ノートブックはエラーなしで完了するまで実行されます。ノートブックでは、**cluster.status()** 関数または **cluster.details()** 関数からの出力は、Ray クラスターが **Active** であることを示します。

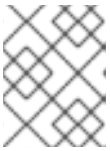
3.2. データサイエンスパイプラインから分散データサイエンスワークロードを実行する

データサイエンスパイプラインから分散データサイエンスワークロードを実行するには、まずパイプラインを更新して Ray クラスターイメージへのリンクを含める必要があります。

前提条件

- **cluster-admin** ロールで OpenShift Container Platform にログインしている。
- 分散ワークロードの設定 の説明に従って分散ワークロードを実行するように設定したデータサイエンスクラスターにアクセスできる。
- 分散ワークロードのクォータ管理の設定 の説明に従って、必要な Kueue リソースを 作成している。
- オプション： 分散ワークロードのクォータ管理の設定 で説明されているように、**LocalQueue** リソースを作成し、次のアノテーションをその **LocalQueue** リソースの設定の詳細に追加して、Ray クラスターの **デフォルト** のローカルキューを 定義している。

```
"kueue.x-k8s.io/default-queue": "true"
```



注記

デフォルトのローカルキューを作成しない場合は、各パイプラインにローカルキューを指定する必要があります。

- S3 互換オブジェクトストレージにアクセスできる。
- Red Hat OpenShift AI にログインしている。
- データサイエンスプロジェクトを作成している。

手順

1. データサイエンスプロジェクトへのデータ接続の追加 の説明に従って、データ接続を作成してオブジェクトストレージをデータサイエンスプロジェクトに接続します。
2. パイプラインサーバーの設定 の説明に従って、データ接続を使用するようにパイプラインサーバーを設定します。
3. 次のようにデータサイエンスパイプラインを作成します。
   a. すべてのパイプラインに必要な **kfp** Python パッケージをインストールします。

```
$ pip install kfp
```

   b. パイプラインに必要な他のすべての依存関係をインストールします。
   c. Python コードでデータサイエンスパイプラインをビルドします。
   たとえば、次の内容を含む、**compile_example.py** という名前のファイルを作成します。

```
from kfp import dsl


@dsl.component(
    base_image="registry.redhat.io/ubi8/python-39:latest",
```

```
    packages_to_install=['codeflare-sdk']
)


def ray_fn(openshift_server: str, openshift_token: str):
    import ray (1)
    from codeflare_sdk import Cluster, ClusterConfiguration, TokenAuthentication,
generate_cert


    auth = TokenAuthentication( (2)
        token=openshift_token, server=openshift_server
    )
    auth_return = auth.login()


    cluster = Cluster( (3)
        ClusterConfiguration(
            name="raytest",
            num_workers=1,
            head_cpus="500m",
            min_memory=1,
            max_memory=1,
            num_gpus=0,
            image="quay.io/project-codeflare/ray:latest-py39-cu118", (4)
            local_queue="local_queue_name", (5)
        )
    )


    print(cluster.status())
    cluster.up() (6)
    cluster.wait_ready() (7)
    print(cluster.status())
    print(cluster.details())


    ray_dashboard_uri = cluster.cluster_dashboard_uri()
    ray_cluster_uri = cluster.cluster_uri()
    print(ray_dashboard_uri, ray_cluster_uri)

    # Enable Ray client to connect to secure Ray cluster that has mTLS enabled
    generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace) (8)
    generate_cert.export_env(cluster.config.name, cluster.config.namespace)


    ray.init(address=ray_cluster_uri)
    print("Ray cluster is up and running: ", ray.is_initialized())


    @ray.remote
    def train_fn(): (9)
        # complex training function
        return 100
```

```
    result = ray.get(train_fn.remote())
    assert 100 == result
    ray.shutdown()
    cluster.down() 10
    auth.logout()
    return result


@dsl.pipeline( 11
    name="Ray Simple Example",
    description="Ray Simple Example",
)


def ray_integration(openshift_server: str, openshift_token: str): 12
    ray_fn(openshift_server=openshift_server, openshift_token=openshift_token)


if __name__ == '__main__':
    from kfp.compiler import Compiler
    Compiler().compile(ray_integration, 'compiled-example.yaml')
```

1 CodeFlare SDK からクラスター機能を定義するパッケージをインポートします。

2 パイプライン実行の作成時に指定した値を使用してクラスターで認証します。クラスターが自己署名証明書を使用する場合は、**TokenAuthentication** パラメーターリストに **ca-cert-path=<path** > を追加します。ここで、<path> は、自己署名証明書を含むクラスター全体の認証局(CA)バンドルへのパスです。

3 **Ray クラスター設定を指定します。これらのサンプルの値を Ray クラスターの値に置き換えます。**

4 **Ray クラスターイメージの場所を指定します。非接続環境を使用している場合は、デフォルト値をご使用の環境の場所に置き換えます。**

5 **Ray クラスターの送信先のローカルキューを指定します。デフォルトのローカルキューを設定した場合は、この行を省略できます。**

6 **指定されたイメージと設定を使用して Ray クラスターを作成します。**

7

Ray クラスターの準備ができるまで待機し、その後続行します。

8

Ray クライアントが相互トランスポート層セキュリティー(mTLS)が有効になっている安全な Ray クラスターに接続できるようにします。mTLS は、OpenShift AI の CodeFlare コンポーネントでデフォルトで有効になっています。

9

このセクションの詳細例を独自のワークロードの詳細に置き換えます。

10

ワークロードが終了したら Ray クラスターを削除します。

11

サンプル名と説明をワークロードの値に置き換えます。

12

Python コードをコンパイルし、出力を YAML ファイルに保存します。

d. Python ファイル (この例では、compile_example.py ファイル) をコンパイルします。

```
$ python compile_example.py
```

このコマンドは YAML ファイル (この例では、compiling-example.yaml) を作成します。これは、次の手順でインポートできます。

4. データサイエンスパイプラインのインポート の説明に従って、データサイエンスパイプラインをインポートします。

5. パイプライン実行のスケジューリング の説明に従って、パイプライン実行をスケジュールします。

6. パイプライン実行が完了したら、トリガーされたパイプライン実行の表示 の説明に従って、トリガーされたパイプライン実行のリストにパイプライン実行が含まれていることを確認

します。

検証

YAML ファイルが作成され、パイプライン実行がエラーなしで完了します。

パイプライン実行の詳細の表示 で説明されているように、実行の詳細を表示できます。

関連情報

- データサイエンスパイプラインの使用
- Ray クラスターのドキュメント

3.3. 非接続環境で分散データサイエンスワークロードを実行する

分散データサイエンスワークロードを非接続環境で実行するには、非接続環境から、Ray クラスターイメージ、ワークロードで使用されるデータセットおよび Python の依存関係にアクセスできる必要があります。

前提条件

- cluster-admin ロールで OpenShift Container Platform にログインしている。
- 非接続のデータサイエンスクラスターにアクセスできる。
- 非接続環境での OpenShift AI Self-Managed のインストールおよびアンインストール の説明に従って、Red Hat OpenShift AI をインストールし、ミラーイメージを作成している。
- 非接続のクラスターから、次のソフトウェアにアクセスできる。
  - Ray クラスターイメージ

- ワークロードで使用されるデータセットとモデル
- Ray イメージまたは非接続のクラスターから利用可能な独自の Python Package Index (PyPI) サーバー内のワークロードの Python 依存関係

- Red Hat OpenShift AI にログインしている。
- データサイエンスプロジェクトを作成している。

手順

1. 分散ワークロードの設定 の説明に従って、切断されたデータサイエンスクラスターが分散ワークロードを実行するように設定します。
2. ノートブックまたはパイプラインの ClusterConfiguration セクションで、image 値が非接続環境からアクセスできる Ray クラスターイメージを指定していることを確認します。
   - ノートブックは、ノートブックの実行時に Ray クラスターイメージを使用して Ray クラスターを作成します。
   - パイプラインは、Ray クラスターイメージを使用して、パイプライン実行中に Ray クラスターを作成します。
3. ワークロードに必要な Python パッケージのいずれかが Ray クラスターで使用できない場合は、プライベート PyPI サーバーから Python パッケージをダウンロードするように Ray クラスターを設定します。

   たとえば、次の例に示すように、Ray クラスターの PIP_INDEX_URL および PIP_TRUSTED_HOST 環境変数を設定して、Python 依存関係の場所を指定します。

```
PIP_INDEX_URL: https://pypi-notebook.apps.mylocation.com/simple
PIP_TRUSTED_HOST: pypi-notebook.apps.mylocation.com
```

   ここでは、以下のようになります。

- PIP_INDEX_URL は、プライベート PyPI サーバーのベース URL を指定します (デフォルト値は https://pypi.org)。

- PIP_TRUSTED_HOST は、ホストが有効な SSL 証明書を持っているか、または安全なチャネルを使用しているかに関係なく、指定されたホストを信頼できるものとしてマークするように Python を設定します。

4. ノートブックからの分散データサイエンスワークロードの実行 または データサイエンスパイプラインからの分散データサイエンスワークロードの実行 の説明に従って、分散データサイエンスワークロードを実行します。

検証

ノートブックまたはパイプライン実行はエラーなしで完了します。

- ノートブックでは、cluster.status() 関数または cluster.details() 関数からの出力は、Ray クラスターが Active であることを示します。

- パイプライン実行の場合は、パイプライン実行の詳細の表示 で説明されているように、実行の詳細を表示できます。

関連情報

- 非接続環境での Red Hat OpenShift AI のインストールおよびアンインストール

- Ray クラスターのドキュメント

第4章 分散ワークロードのモニタリング

OpenShift AI では、分散ワークロードのプロジェクトメトリクスを表示し、選択したプロジェクト内のすべての分散ワークロードのステータスを表示できます。これらのメトリクスを使用して、分散ワークロードで使用されるリソースを監視し、プロジェクトリソースが正しく割り当てられているかどうかを評価し、分散ワークロードの進捗を追跡し、必要に応じて修正アクションを特定できます。

4.1. 分散ワークロードのプロジェクトメトリクスの表示

OpenShift AI では、分散ワークロードの次のプロジェクトメトリクスを表示できます。

- CPU - 選択したプロジェクトのすべての分散ワークロードで現在使用されている CPU コアの数。
- メモリー - 選択したプロジェクトのすべての分散ワークロードによって現在使用されているメモリーの容量(GiB 単位)。

これらのメトリクスを使用して、分散ワークロードで使用されるリソースを監視し、プロジェクトリソースが適切に割り当てられているかどうかを評価できます。

前提条件

- Red Hat OpenShift AI をインストールしている。
- OpenShift AI がインストールされている OpenShift クラスターで、ユーザーのワークロードモニタリングが有効化されている。
- OpenShift AI にログインしている。
- 特殊な OpenShift AI グループを使用している場合は、OpenShift のユーザーグループ、または、管理者グループ (rhoai-users、rhoai-admins など) に属している。
- データサイエンスプロジェクトには分散ワークロードが含まれています。

手順

1. OpenShift AI の左側のナビゲーションペインで、Distributed Workloads Metrics をクリックします。

2. プロジェクト リスト から、監視する分散ワークロードを含むプロジェクトを選択します。

3. Project metrics タブをクリックします。

4. オプション： Refresh interval リストから値を選択し、メトリクスページのグラフの更新頻度を指定して最新のデータを表示します。

   15 秒、30 秒、1 分、5 分、15 分、30 分、1 時間、2 時間、1 日の値のいずれかを選択できます。

5. Requested resources セクションで、CPU と Memory グラフを確認し、分散ワークロードによって要求されたリソースを以下のように特定します。

   - 選択したプロジェクトによって要求されます。
   - アクセス権できない選択されたプロジェクトおよびプロジェクトを含むすべてのプロジェクトで要求されます。
   - クラスターキューによって提供されるすべてのプロジェクトの共有クォータの合計

   各リソースタイプ(CPU および メモリー)について、Total shared quota 値から Requested をすべてのプロジェクト の値で減算し、そのリソースクォータが要求されておらず、すべてのプロジェクトで使用できる量を計算します。

6. Top resource-consuming distributed workloads セクションまで下にスクロールして、次のグラフを確認します。

   - 最も多くの CPU リソースを消費している、最大 5 個の分散ワークロード

- メモリーを消費する上位 5 個の分散ワークロード

それぞれのケースで、使用されている CPU またはメモリーの量を特定することもできます。

7. スクロールダウンして、選択したプロジェクト内のすべての分散ワークロードを一覧表示する Distributed workload resource metrics テーブルを表示し、現在のリソースの使用状況と各分散ワークロードのステータスを示します。

各テーブルエントリーで、進行状況バーは、要求された CPU とメモリーが現在この分散ワークロードによって使用されている量を示します。CPU の実際の使用量および（コアで測定）およびメモリー(GiB で測定)の実際の使用量の数値を表示するには、各進捗バーにカーソルを置きます。実際の使用量と要求された使用量を比較して、分散されたワークロード設定を評価します。必要に応じて、分散ワークロードを再設定して、要求されたリソースを減らまたは増やします。

検証

Project metrics タブでは、グラフと表は選択したプロジェクトで分散されたワークロードのリソース使用率データを提供します。

4.2. 分散ワークロードのステータスの表示

OpenShift AI では、選択したプロジェクト内のすべての分散ワークロードのステータスを表示できます。分散ワークロードの進捗を追跡し、必要に応じて修正アクションを特定できます。

前提条件

- Red Hat OpenShift AI をインストールしている。
- OpenShift AI がインストールされている OpenShift クラスターで、ユーザーのワークロードモニタリングが有効化されている。
- OpenShift AI にログインしている。
- 特殊な OpenShift AI グループを使用している場合は、OpenShift のユーザーグループ、または、管理者グループ (rhoai-users、rhoai-admins など) に属している。

- データサイエンスプロジェクトには分散ワークロードが含まれています。

手順

1. OpenShift AI の左側のナビゲーションペインで、Distributed Workloads Metrics をクリックします。
2. プロジェクト リスト から、監視する分散ワークロードを含むプロジェクトを選択します。
3. Distributed workload status タブをクリックします。
4. オプション： Refresh interval リストから値を選択し、メトリクスページのグラフの更新頻度を指定して最新のデータを表示します。

   15 秒、30 秒、1 分、5 分、15 分、30 分、1 時間、2 時間、1 日の値のいずれかを選択できます。
5. ステータスの 概要 セクション で、選択したプロジェクト内のすべての分散ワークロードのステータスの概要を確認します。

   ステータスは、Pending、In admisible、Admitted、Running、Evicted、Succeed、または Failed のいずれ かになります。
6. スクロールダウンして Distributed workloads テーブルを表示します。このテーブルには、選択したプロジェクトの分散ワークロードがすべて一覧表示されます。この表は、分散ワークロードごとに優先順位、ステータス、作成日、最新のメッセージを提供します。

   最新のメッセージは、分散ワークロードの現在のステータスに関する詳細情報を提供します。最新メッセージを確認し、必要な修正措置を特定します。たとえば、要求されるリソースが利用可能なリソースを超えるため、分散されたワークロードは Inadmissible になる可能性があります。このような場合は、分散ワークロードを再設定して要求されたリソースを減らしたり、プロジェクトのクラスターキューを再設定してリソースクォータを増やすこともできます。

検証

Distributed workload status タブでは、選択したプロジェクト内のすべての分散ワークロードのステータスをまとめたグラフが表示され、各分散ワークロードのステータスについての詳細が提供されま

す。