Alibaba Cloud Container Service

ディープラーニング

Document Version20190701

目次

1 Swarm クラスター

2 概要

Alibaba Cloud のパワフルなコンピューティング機能をベースにしたディープラーニングソリューションは、簡単でオープンなエンドツーエンドディープラーニングサービスプラットフォームを提供しています。 このソリューションにより、データ科学者およびアルゴリズムエンジニアは、Alibaba Cloud リソース (ECS (Elastic Compute Service) インスタンス、GPU インスタンス、 Alibaba Cloud HPC、OSS (Object Storage Service)、 Elastic MapReduce および Server Load Balancer を含む) を使用して、データ準備、モデル開発、モデルトレーニング、評価、予測、およびその他のタスクを迅速に実行できます。 このソリューションにより、ディープラーニング機能の API へ転送が簡単になり、業務アプリケーションとの統合が促進されます。

ディープラーニングソリューションには、以下の特長があります。

- シンプル: ディープラーニングプラットフォームの構築および管理のためのしきい値を下げます。
- 効率的: CPU や GPU などの異種のコンピューティングリソースの効率を向上し、統合されたユーザーエクスペリエンスが提供されます。
- オープン: TensorSlow、Keras および MXNet などの複数のメインストリームディープラーニングフレームワークをサポートし、カスタマイズ環境がサポートされます。
- フルサイクル: Alibaba Cloud のパワフルなサービスシステムに基づいたエンドツーエンドのディープラーニングタスクプロセスを構築するためのベストプラクティスが提供されます。
- サービス指向: ディープラーニング機能をサービスに変換し、クラウド上のアプリケーションと簡単に統合します。

はじめに

1. 環境を準備します。

   コンテナークラスターの作成. OSS データボリュームを使用して、 データを保存します。*OSSFS データボリュームの作成*

2. *Jupyter 環境の作成Jupyter 環境における Git を利用したコード管理*をご参照ください。
3. スタンドアロンモデルトレーニングまたは分散モデルトレーニングを実行して、モデルをエクスポートします。
4. エクスポートされたモデルを利用して、*TensorFlow Serving の利用*を行います。

3 環境の準備

3.1 データボリュームの作成

OSSFS は FUSE をベースにしたファイルシステムで、Alibaba Cloud により公式に提供されています (プロジェクトのホームページは、https://github.com/aliyun/ossfsをクリックしご参照ください)。 OSSFS データボリュームは、OSS (Object Storage Service) バケットをデータボリュームとして パッケージ化できます。

OSSFS のパフォーマンスおよび機能は、ローカルファイルシステムのパフォーマンスおよび機能とは異なります。ネットワークを通じてデータをクラウドと同期させる必要があるためです。OSSFS のログなどのファイルを絶えず書き換える必要があるデータベースやアプリケーションなど、 I/O 集約型アプリケーションは実行しないことを推奨します。 OSSFS は、コンテナー間の設定ファイルの共有や、書き換えを必要としない添付ファイルのアップロードなどのシナリオに適用できます。

OSSFS は、以下の点でローカルファイルシステムとは異なります。

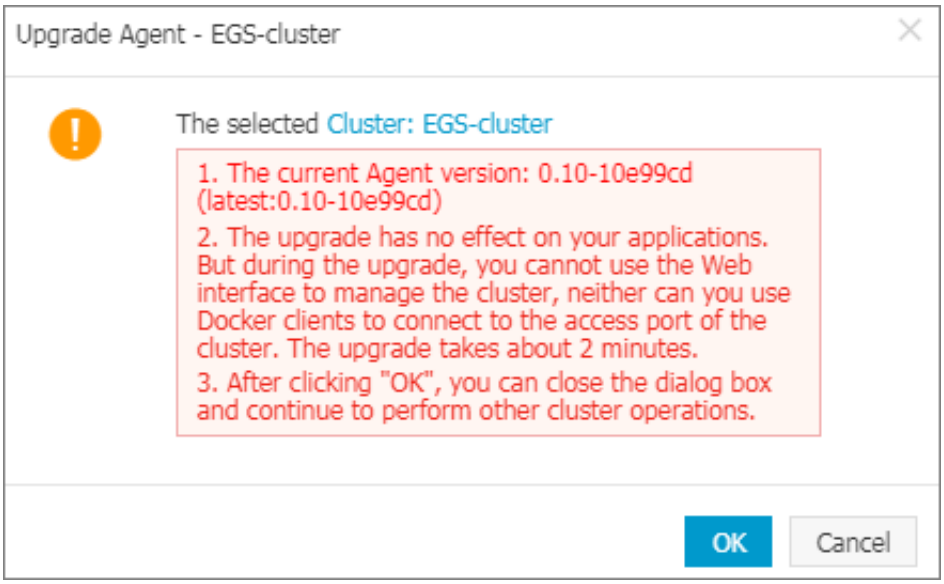
- ランダム書き込みや追加書き込みにより、ファイル全体が上書きされます。
- システムがリモートで OSS サーバーへアクセスする必要があるため、リストディレクトリなどのメタデータ操作のパフォーマンスが低下します。
- ファイルやフォルダーの名称変更操作はアトミックではありません。
- 複数のクライアントが同じ OSS バケットに接続されている場合、各クライアントの動作を自ら調整します。 たとえば、複数のクライアントによる同じファイルへの書き込みを避けます。
- ハードリンクはサポートされません。

前提条件

データボリューム機能を有効化するために、クラスターは次の 2 つの条件を満たす必要があります。

- クラスターエージェントのバージョンが 0.6 以降であること。

  クラスターリストのページでエージェントのバージョンを参照できます。 [詳細] > [エージェントのアップグレード] をクリックします。



  エージェントのバージョンが 0.6 以前の場合、エージェントをアップグレードします。 エージェントのアップグレード方法について詳しくは、エージェントのアップグレードをご参照ください。

- クラスターに acsvolumedriver アプリケーションがデプロイされていること。 最新バージョンへのアップグレードを推奨します。

  システムサービスのアップグレードにより、acsvolumedriver アプリケーションのデプロイおよびアップグレードができます。 詳しくは、システムサービスのアップグレードをご参照ください。

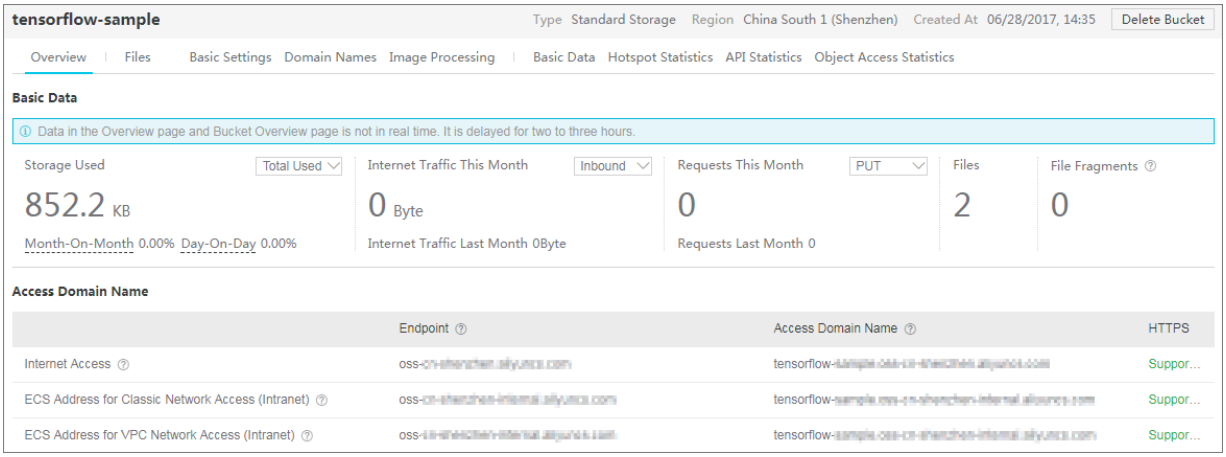
  > 注 :
  > acsvolumedriver がアップグレードまたは再起動された場合、使用中の OSSFS データボリュームが再起動され、お使いのサービスも再起動されます。

手順

手順 1: OSS バケットの作成

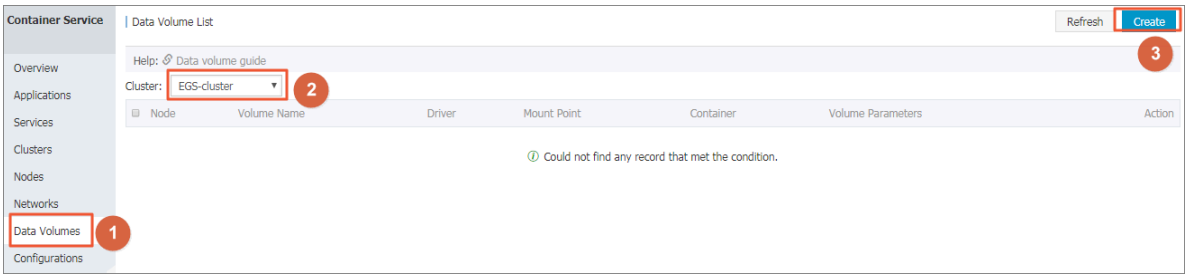
*OSS コンソール*にログインして、バケットを作成します。 詳しくは、バケットの作成をご参照ください。

この例では、中国 (深セン) に配置されるバケットが作成されます。



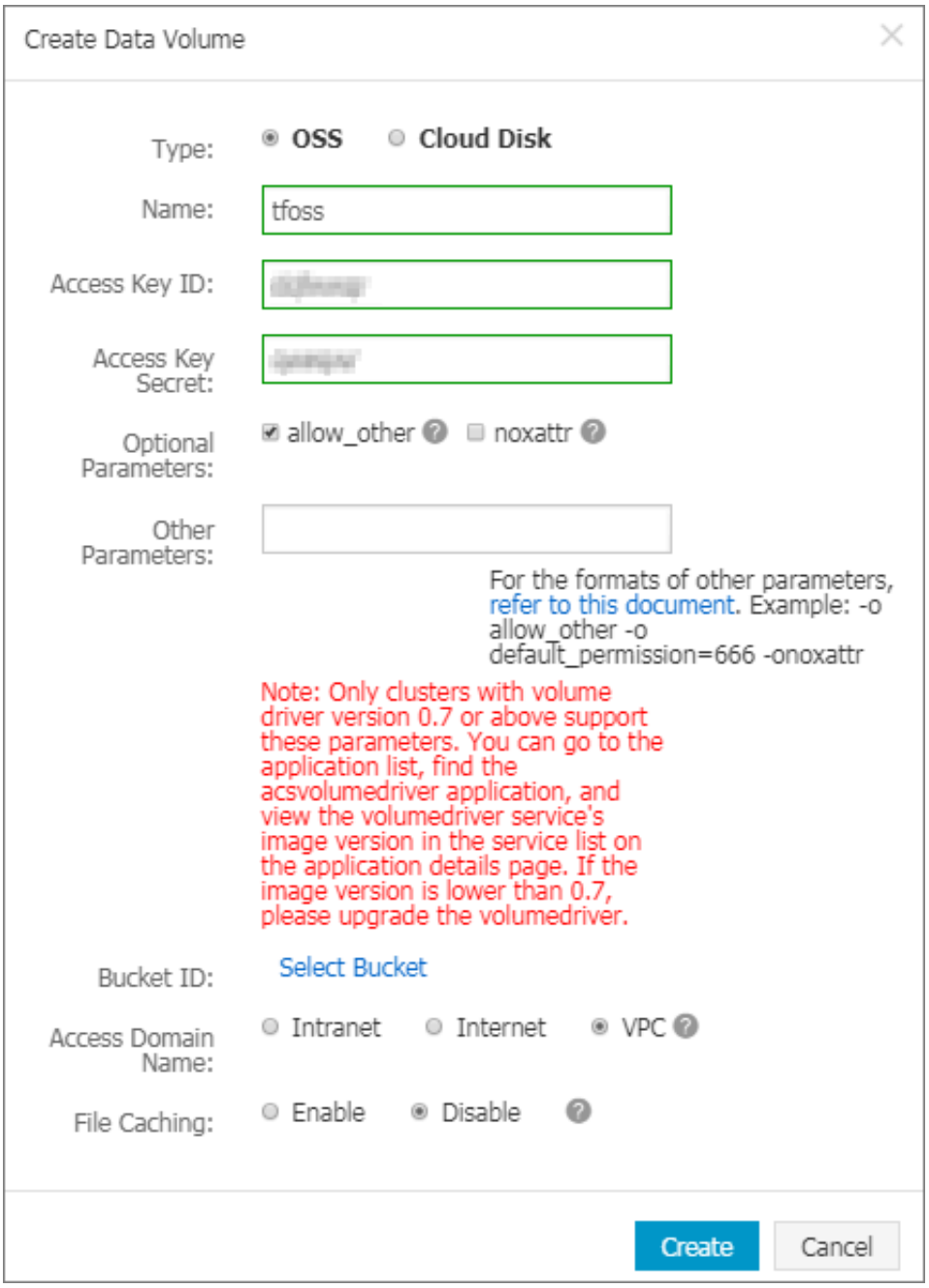
手順 2: OSSFS データボリュームの作成

1. *Container Service* コンソールにログインします。
2. 左側のナビゲーションウィンドウから [データボリューム] をクリックします。
3. [クラスター] ドロップダウンリストから、データボリューム (この例では、"tfoss") を作成するクラスターを選択します。 右上の [作成] をクリックします。



4. [データボリュームの作成] ダイアログボックスが表示されます。 [データボリュームタイプ] として [OSS] を選択します。データボリュームパラメーターを設定して、[作成] をクリックしま

す。 Container Service により、クラスターのすべてのノードに同名のデータボリュームが作成されます。



- 名前: データボリューム名はクラスターで一意でなくてはなりません。
- Access Key ID/Access Key シークレット: OSS へのアクセスに必要な Access Key です。 *Access Key コンソール*から取得できます。
- バケット ID: 使用する OSS バケットの名称です。 [バケットの選択] をクリックします。 表示されたダイアログボックスでバケット (この例では、"tensorflow-sample") を選択して [選択] をクリックします。
- アクセスドメイン名: [VPC] を選択します。

- ファイルキャッシュ: 複数のマシン上にある同一ファイルの変更を同期させる場合は [無効化] を選択します (たとえば、マシン A 上のファイルを変更して、マシン B で変更した内容を読み込む場合)。

  > 注:
  > 同じフォルダにたくさんのファイルがある場合、ファイルキャッシュの無効化によりフォルダの "ls" が特に遅くなります。 前述の要件がない場合、ファイルキャッシュを有効化して "ls" を高速化します。

その後の操作

データボリューム作成後、アプリケーションで利用できます。 アプリケーションでのデータボリュームの使用方法については、サードパーティのデータボリュームの使用をご参照ください。

3.2 コンテナークラスターの作成

ディープラーニングソリューションは、ECS (Elastic Compute Service) インスタンスまたは GPU インスタンスを備えたコンテナークラスターをサポートしています。 このドキュメントでは、例として GPU インスタンスを備えたコンテナークラスターを使用します。

> 注:
> ECS インスタンスを備えたコンテナークラスターの作成方法については、クラスターの作成をご参照ください。

制限

- 現在、Container Service は以下のリージョンにおいて GN4 GPU インスタンスを備えたクラスターの作成をサポートしています: 中国 (深セン)、中国 (上海)、中国 (北京) および米国 (シリコンバレー)。
- 現在、GN4 GPU インスタンスは VPC (Virtual Private Cloud) のみのサポートです。

前提条件

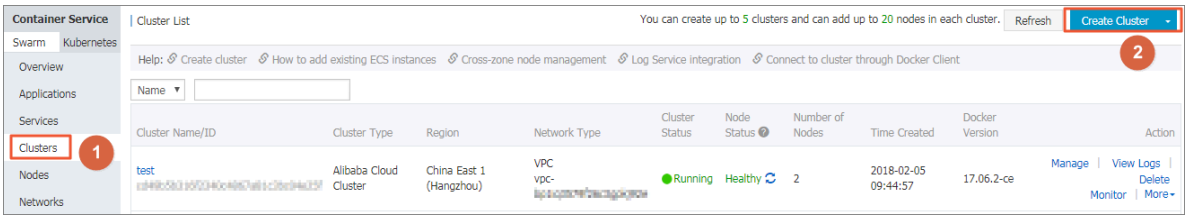
現在、従量課金 GPU Compute タイプ GN4 インスタンスを有効化する必要があります。 有効化には以下のように、*ECS チケットを起票し、サポートセンターへお問い合わせください*。

「従量課金 GPU Compute タイプ GN4 インスタンスを有効化を希望します。 よろしくお願いします。」

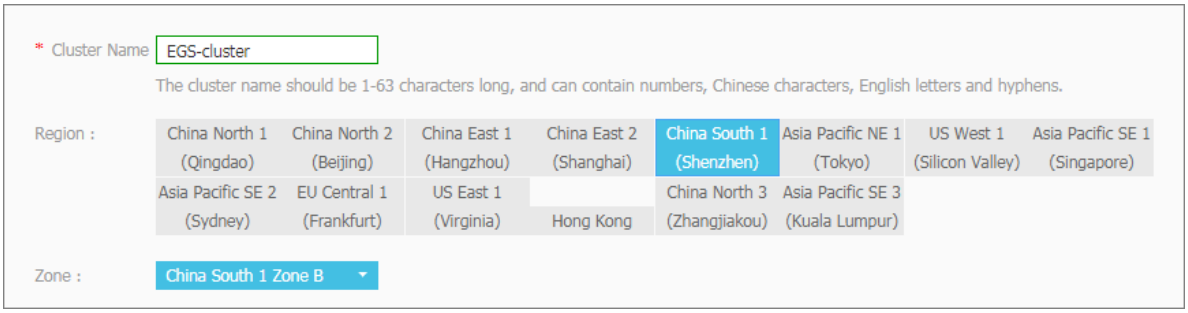
手順

1. *Container Service コンソール*にログインします。

2. 左側のナビゲーションウィンドウから [Swarm] > [クラスター] をクリックします。 右上にある [クラスターの作成] をクリックします。



3. 以下の設定を完了させます。 この例では、"EGS-cluster" という名前を設定したクラスターを中国 (深セン) リージョンに作成します。



- クラスター名: 作成するクラスター名です。 1 文字以上 64 文字以内で、数字、漢字、英字およびハイフン (-) を含むことができます。

  注 :
  クラスター名は、同じアカウント、同じリージョン内で一意である必要があります。

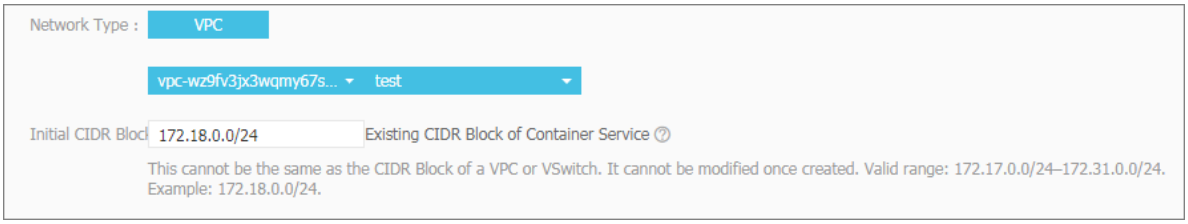
- リージョン: クラスターをデプロイするリージョンを選択します。 [中国 (深セン)] 、[中国 (上海)]、[中国 (北京)] または [米国 (シリコンバレー)] を選択します。

  注 :
  現在、Container Service では以下のリージョンにおいて GN4 GPU インスタンスを持つクラスターの作成がサポートされます: 中国 (深セン)、中国 (上海)、中国 (北京) および米国 (シリコンバレー)。

- ゾーン: クラスターのゾーンを選択します。

  注 :
  サーバーの配分に応じてリージョンおよびゾーンを選択できます。

4. [ネットワークタイプ] として [VPC] を選択して、設定を完了させます。

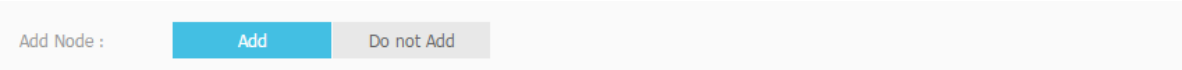


VPC により、Alibaba Cloud をベースにした独立したネットワーク環境を構築できます。 独自の仮想ネットワークの完全な制御が可能です。 これには、フリー IP アドレス範囲、CIDR (Classless Inter-Domain Routing) ブロックの分割、ルートテーブルとゲートウェイの設定が含まれます。

VPC、VSwitchId およびコンテナーの最初の CIDR ブロック (Docker コンテナーが属するサブネット CIDR ブロック)を 指定します。 IP の管理を容易にするため、それぞれの仮想マシンコンテナーは複数の CIDR ブロックに属し、 コンテナーサブネット CIDR ブロックは仮想マシンの CIDR ブロックと競合できません。

ネットワークの競合のような問題を避けるために、お使いの独自の VPC と VSwitchId をコンテナークラスターに対して 構築することを推奨します。

5. ノードを追加するかどうかを選択します。



新しいインスタンスを複数持つクラスターを作成することも、ゼロノードクラスターの作成後に既存のインスタンスを追加することもできます。 既存のインスタンスのクラスターへの追加方法については、「既存のインスタンスの追加」をご参照ください。

・ 追加する場合

a. ノードのオペレーティングシステムを選択します。

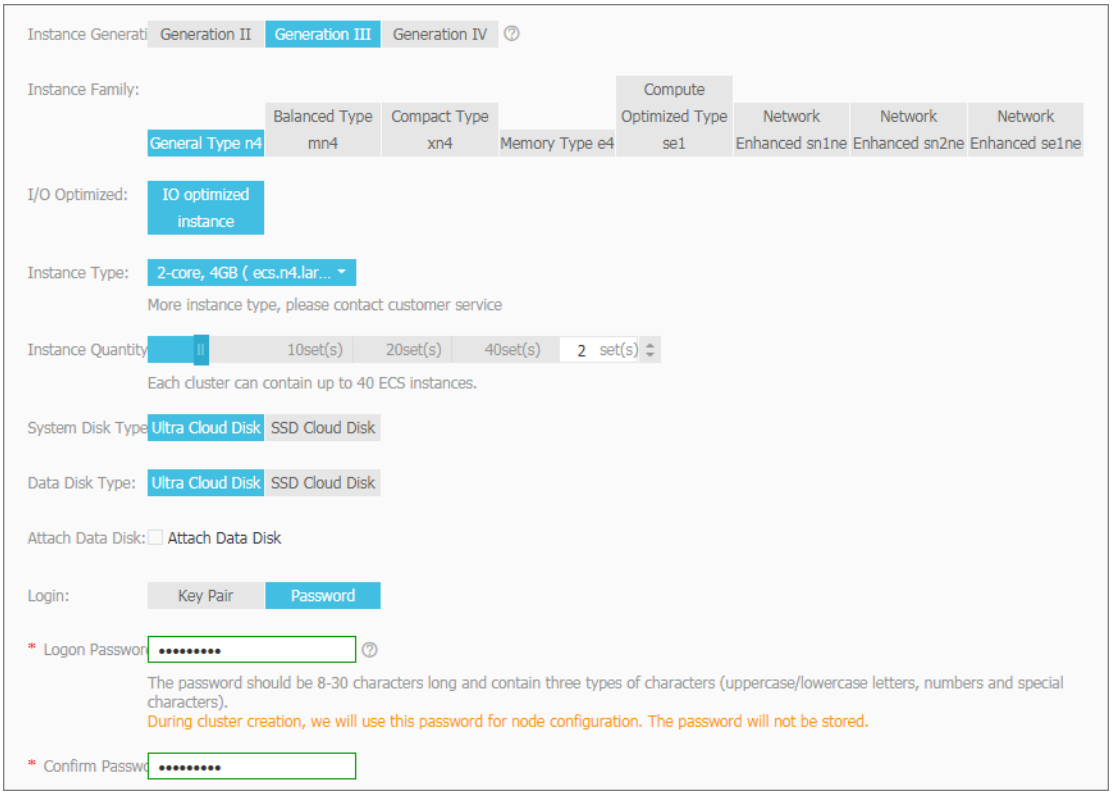
現在、サポートされるオペレーティングシステムは Ubuntu 16.04 64bit および CentOS 7.4 64bit です。

b. インスタンスの仕様を設定します。

- [第 3 世代] を [インスタンスの世代] として、[GPU Compute タイプ gn4] を [インスタンスファミリー] として選択します。
- [32 コア、48 GB (ecs.gn4.8xlarge)] または [56 コア、96 GB (ecs.gn4.14xlarge)] を [インスタンスタイプ] として選択します。

注 :

GN4 GPU インスタンスの使用が承認されていても、 この 2 つのインスタンスタイプが見つからないのは、この 2 つのインスタンスに対して利用できるリソースがないためです。 インスタンスを後で購入するか、翌日に購入することを推奨します。



インスタンス数、データディスク容量 (GPU インスタンスはデフォルトで 20 GB のシステムディスクを持ちます) およびログインパスワードを設定できます。

注：

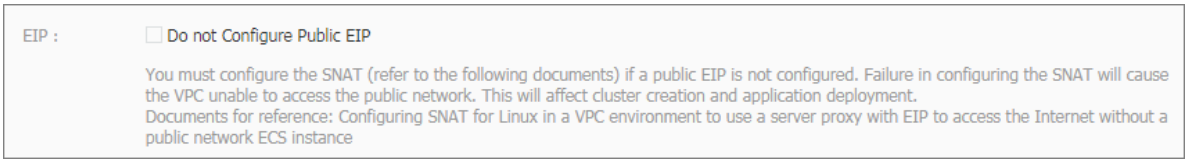
- [データディスクの接続] チェックボックスがオンの場合、データディスクは `/ var / lib / docker` ディレクトリに接続され、 Docker イメージおよびコンテナーのストレージとして使用されます。
- パフォーマンスと管理の点から、独立したデータディスクをホストに接続し、Dockert ボリュームを利用してコンテナーの永続データを管理することを推奨します。

・ 追加しない場合

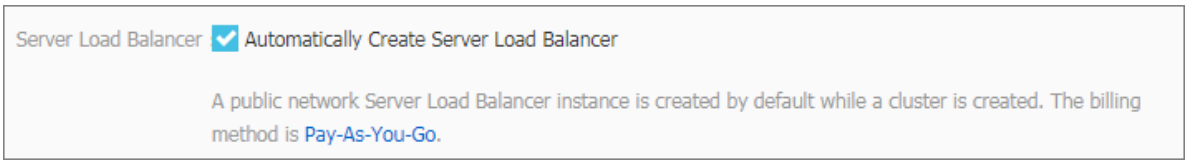
[既存のインスタンスの追加] をクリックして既存のインスタンスをクラスターに追加、または、 クラスター作成後の クラスターリストページで [既存のインスタンスの追加] をクリックして既存のインスタンスをクラスターに追加できます。

6. パブリック EIP (Elastic IP) を設定するかどうかを選択します。

   ネットワークタイプとして VPC を選択した場合、Container Service はデフォルトで各インスタンスに対して *EIP* を設定します。 インスタンスの設定が必要ない場合、[パブリック EIP を設定しない] チェックボックスをオンにして、 SNAT ゲートウェイを設定します。



7. Server Load Balancer インスタンスを作成するかどうかを選択します。

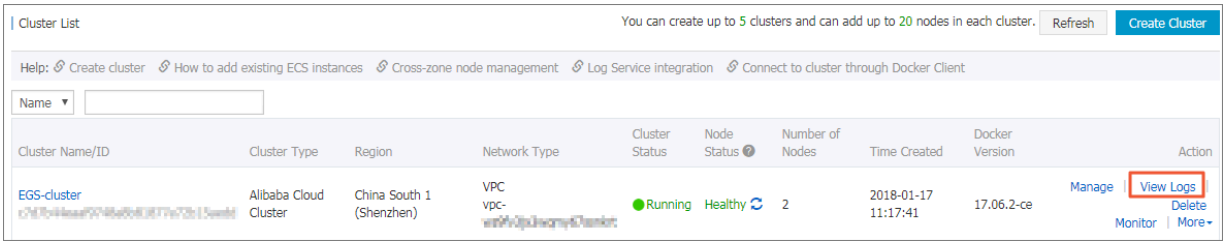


   デフォルトで、[Server Load Balancer の自動作成] チェックボックスがオンになっています。 チェックボックスがオンの場合、クラスター作成後にインターネット Server Load Balancer インスタンスが作成されます。 クラスターのアプリケーションに Server Load Balancer インスタンスを利用してアクセスできます。 これは、従量課金 Server Load Balancer インスタンスです。

8. [クラスターの作成] をクリックします。

その後の操作

クラスターリストページで、クラスターの右側にある [ログの表示] をクリックし、クラスターの作成処理のログを 表示できます。



4 モデル開発

4.1 Jupyter 環境の作成

前提条件

モデルトレーニングタスクの実行前に、以下の操作を実行したことを確認します。

- 一定数のエラスティックコンピューティングリソース (ECS (Elastic Compute Service) または EGS) を含むコンテナークラスターを作成します。 詳しくは、コンテナークラスターの作成をご参照ください。
- モデルトレーニング用データの保存に OSS (Object Storage Service) を使用するために、同じアカウントを使用して OSS バケットを作成します。 前述のコンテナークラスターにデータボリュームを作成し、OSS バケットをローカルディレクトリとして、トレーニングタスクを実行するコンテナーにマウントします。 詳しくは、データボリュームの作成をご参照ください。

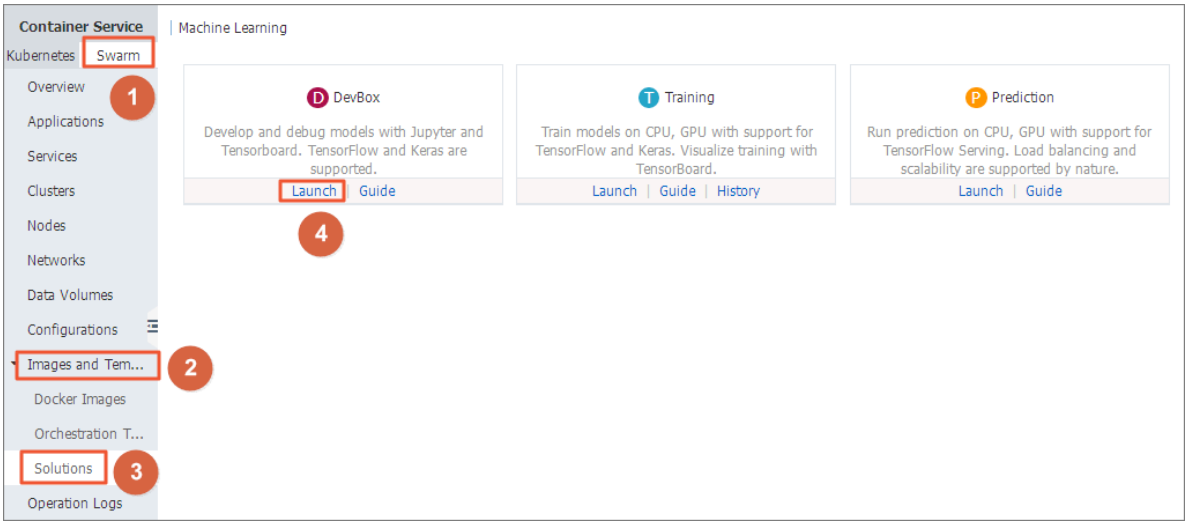
規則

アプリケーションのコードでトレーニングデータの読み込み、およびトレーニングデータの出力を簡単にするため、トレーニングボリューム上のデータは `/input` ディレクトリに 保存されます。 コードはこのディレクトリからデータを読み込みます。

手順

1. *Container Service* コンソールにログインします。
2. 左側のナビゲーションウィンドウから [Swarm] > [イメージとテンプレート] > [ソリューション] をクリックします。

3. [DevBox] で [起動] をクリックします。

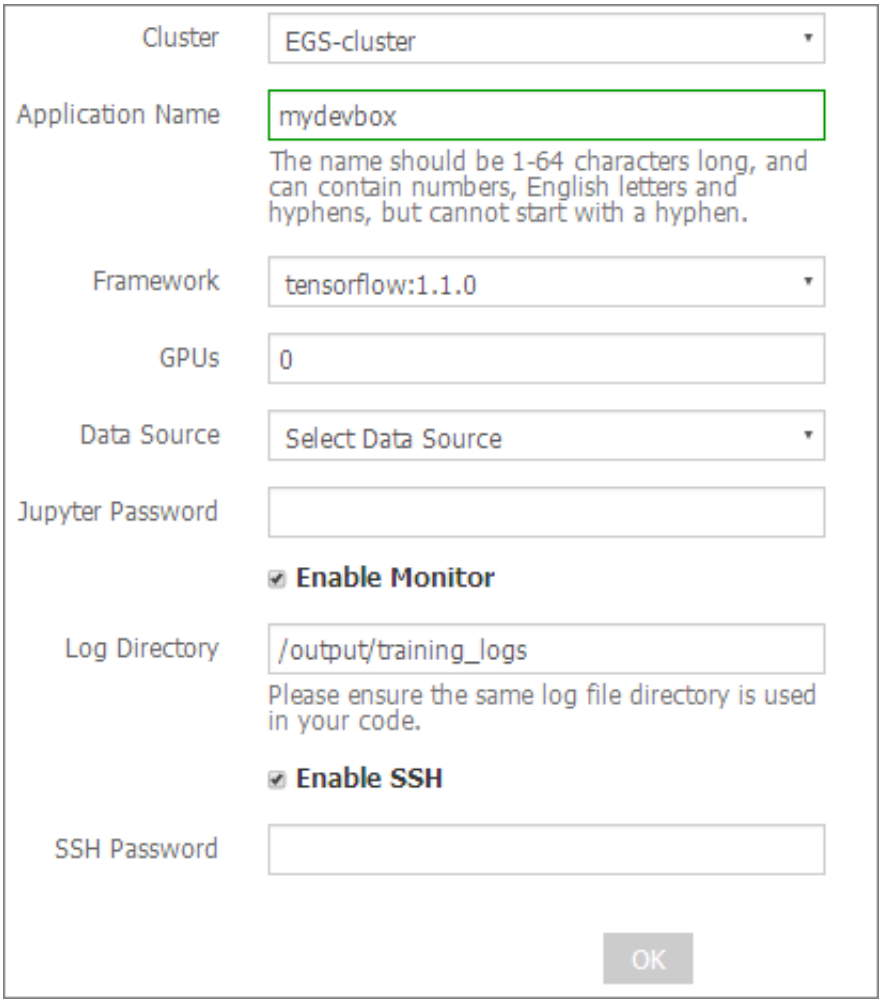


4. Jupyter 環境を作成するための基本情報を設定します。

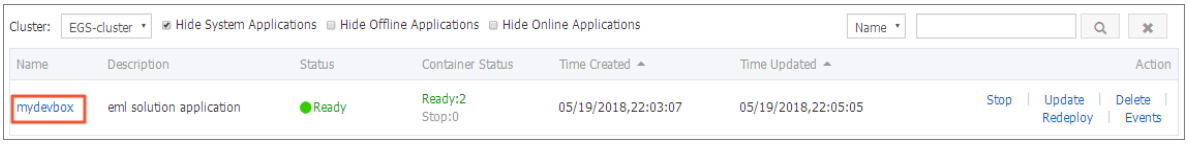
- クラスター: 作成したモデルデプロイアプリケーションをデプロイするクラスターを選択します。 この例では、"EGS-cluster" を選択します。
- アプリケーション名: 作成するアプリケーションの名前です。 1 文字以上 64 文字以内で、数字、英字およびハイフン (-) を含むことができますが、ハイフン (-) で始めることはできません。
- フレームワーク: サポートされるフレームワークには、TensorFlow、Keras および Python が含まれます。
- GPU: 使用する GPU 数です。 このフィールドで "0" を設定すると、GPU は使用されません。
- データソース: トレーニングデータを保存するデータソースを選択します。 OSS によりクラスター上の作成されたデータボリュームの選択、または [ローカルディレクトリ] を選択して絶対パスを入力します。 [データソースなし] を選択することもできます。
- Jupyter パスワード: Jupyter へのログインに使用するパスワードです。
- モニターの有効化: トレーニングステータスのモニタリングのために、 TensorBoard を使用するかどうかを選択します。 このチェックボックスをオンにした場合、[ログディレクトリ] フィールドにトレーニングログのパスを入力して、入力したパスがトレーニングコード内のログ出力パスと同じであることを確認します。
- SSH の有効化: SSH によるサービスへのアクセスを許可するかを選択します。 このチェックボックスをオンにした場合、 SSH パスワードを入力します。

 注 :

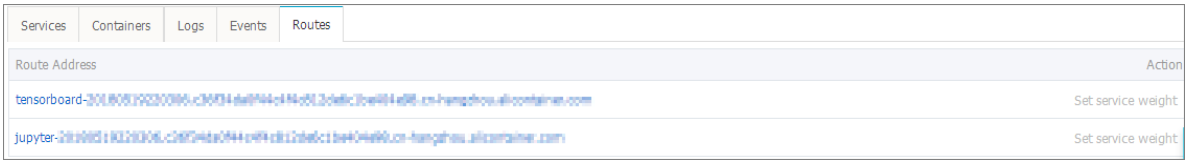
SSH によるサービスへのアクセス方法については、*SSH を利用した Jupyter サービスへのアクセス*をご参照ください。



5. 設定完了後、[OK] をクリックします。
6. [アプリケーションリスト] ページで作成したアプリケーションの名前をクリックします。

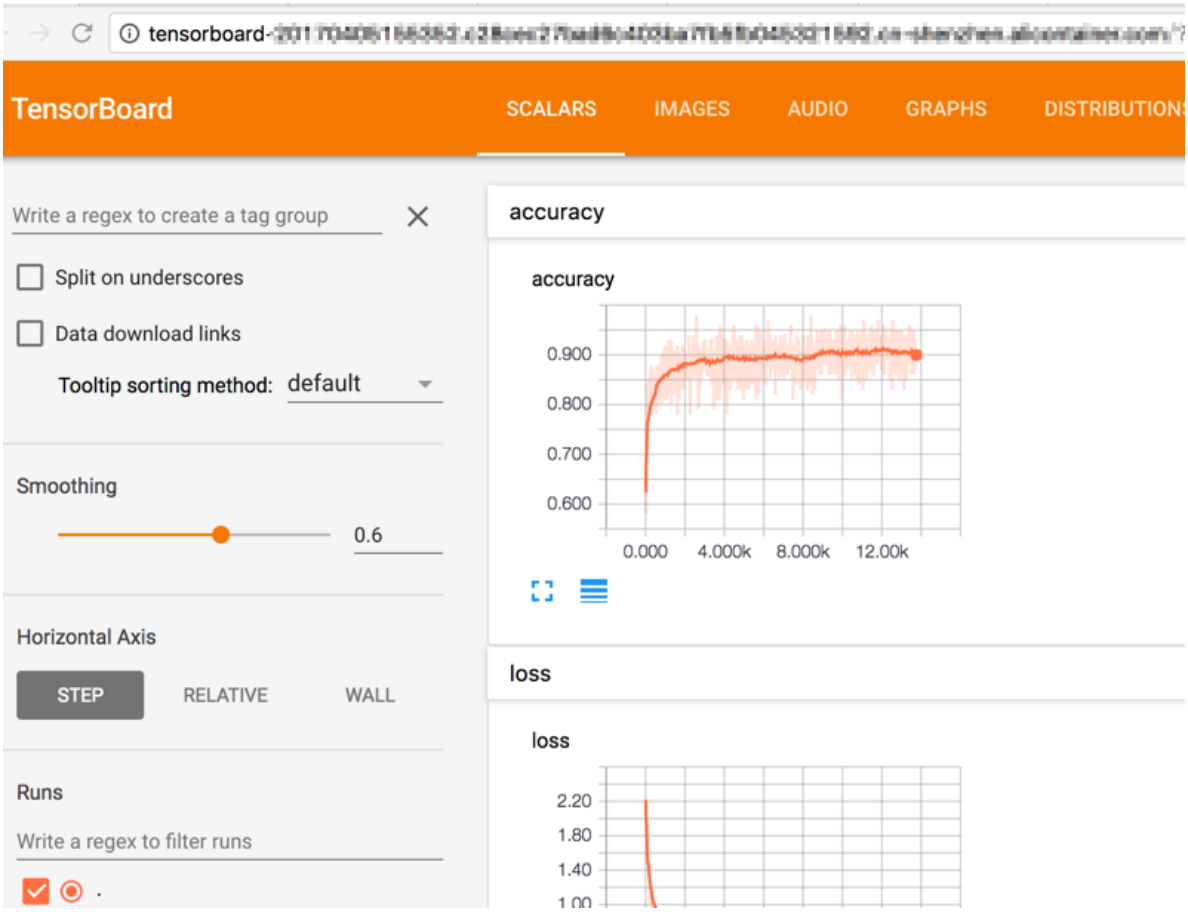


7. [ルート] タブをクリックします。 "jupyter" と "tensorboard" で始まる 2 つのリンクが表示されます。



8. "jupyter" で始まるリンクをクリックし、Jupyter のパスワードを入力して Jupyter 環境にアクセスします。

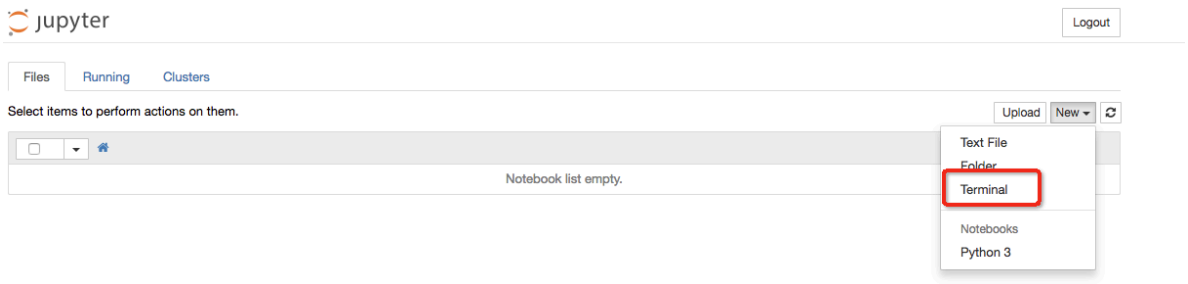
9. "tensorboard" で始まるリンクをクリックして、トレーニング結果を参照します。



10.分散ストレージ上のトレーニングデータは、ローカルの /input フォルダに保存されます。 このフォルダからデータを読み込むことができます。

4.2 Jupyter 環境における Git を利用したコード管理

1. Jupyter のホームページでターミナルを作成します。



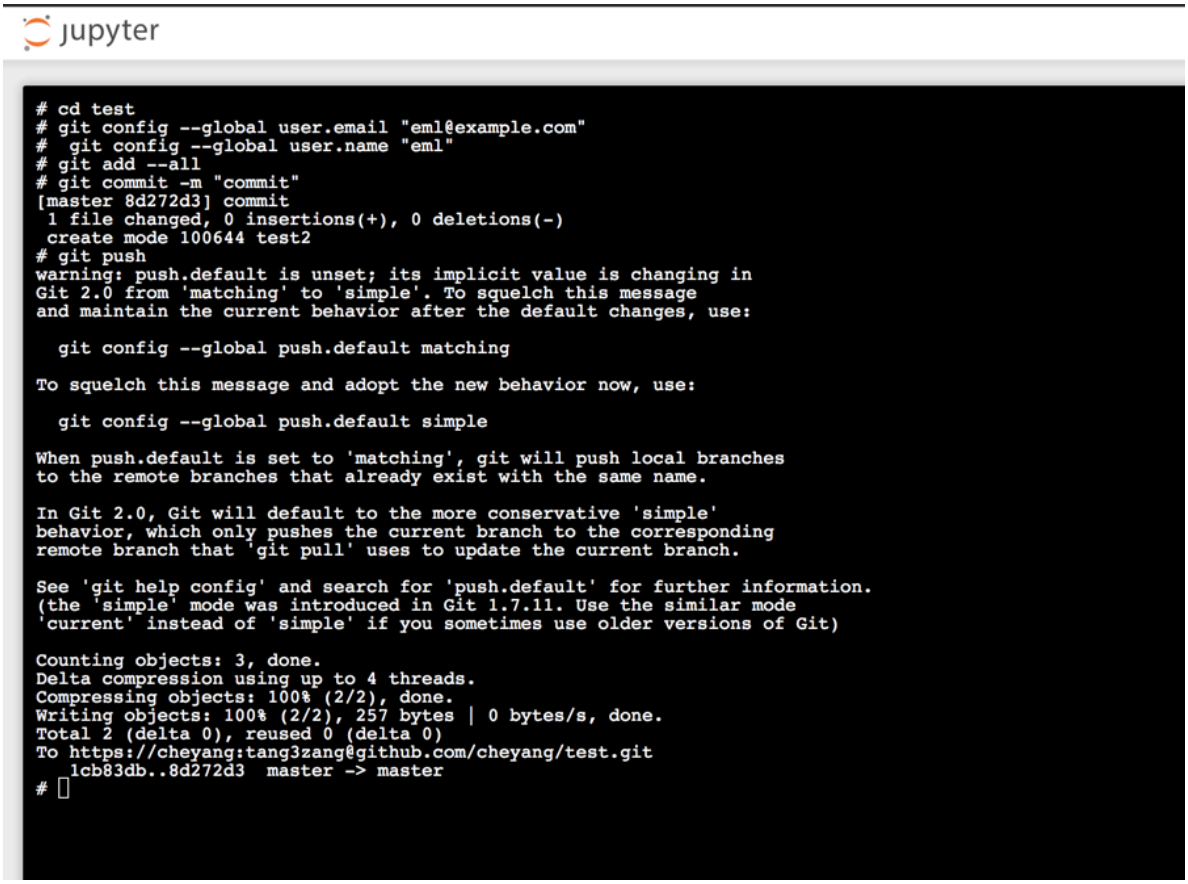
2. ターミナルで `git clone` を実行して、アプリケーションのコードをダウンロードします。

```
git clone https://{id}:{password}@github.com/{id}/test.git
Cloning into 'test'...
remote: Counting objects: 3, done.
```

```
remote :  Total   3  ( delta   0 ),  reused   0  ( delta   0 ),
pack - reused   0
Unpacking   objects :  100 % ( 3 / 3 ),  done .
Checking   connectivi  ty ...  done .
```

`{ id }` および `{ password }` はそれぞれ、GitHub のユーザー名およびパスワードを 示しています。

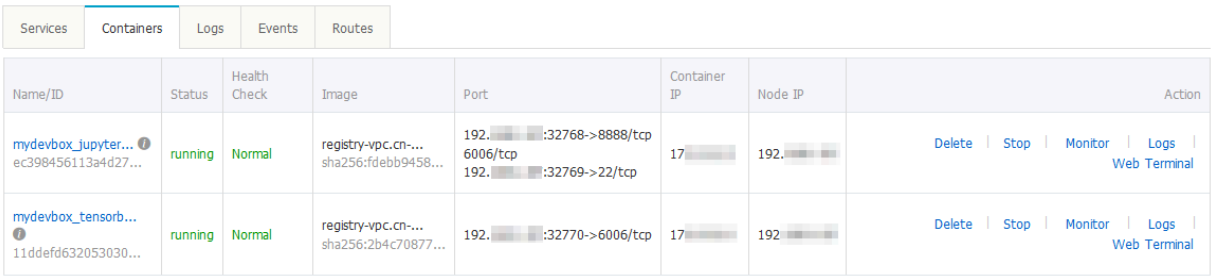
3. Jupyter のホームページに戻ります。 表示されたアプリケーションのコードを確認し、Jupyter を使用して対応するコードを開発できます。
4. ターミナルに戻り、Git を使用してコードを送信できます。



4.3 SSH を利用した Jupyter サービスへのアクセス

Jupyter 環境の作成の実行時に [SSH の有効化] チェックボックスをオンにした場合、 このドキュメントの手順に従い、SSH を利用して Jupyter サービスにアクセスできます。

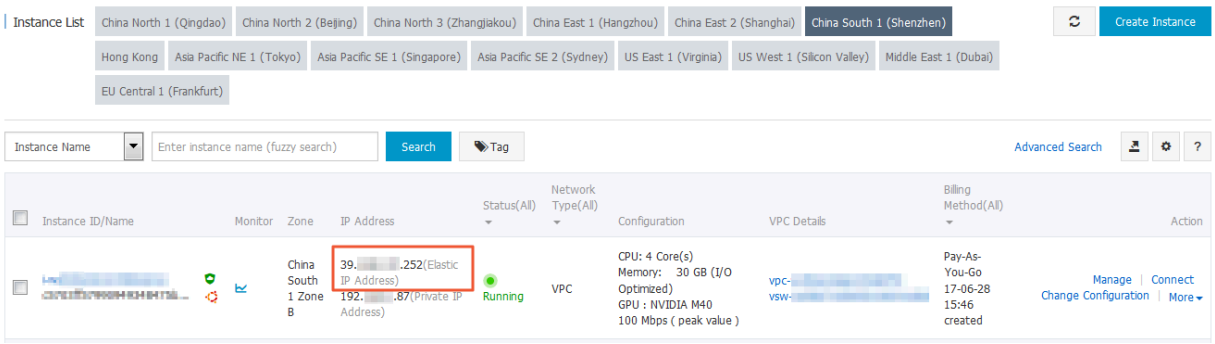
この例では、SSH アクセス用のポートマッピングは 192. 32769->22/tcp です。 192. ** は ECS (Elastic Compute Service) インスタンスのプライベート IP アドレスです。 Jupyter サービスへのアクセスには、 この IP アドレスは使用できません。 Jupyter サービスにアクセスするために、ECS インスタンスのパブリック EIP (Elastic IP) を使用して外部からこれらのサービスに SSH アクセスします。



ECS コンソール にログインして、ECS インスタンスにバインドされているパブリック EIP を確認できます。 このページの例では、パブリック EIP は 39. 252 となります。

注：

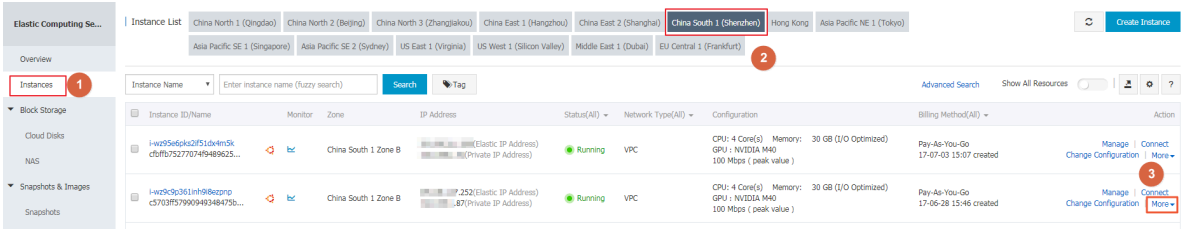
ECS インスタンスにパブリック EIP がバインドされていない場合、ECS インスタンスにパブリック IP をバインドします。 詳しくは、「*パブリック EIP アドレスの管理*」をご参照ください。



さらに、ECS インスタンスのパブリック EIP を使用して外部から Jupyter に SSH アクセスするために、最初にセキュリティグループルールを設定し、32769 ポートを開きます。

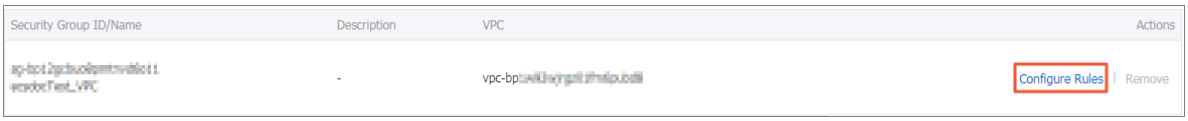
セキュリティグループルールの設定

1. *ECS コンソール*にログインします。
2. 左側のナビゲーションウィンドウから [インスタンス] をクリックします。 リージョンを選択します (このページの例では、[中国 (深セン)]となります)。

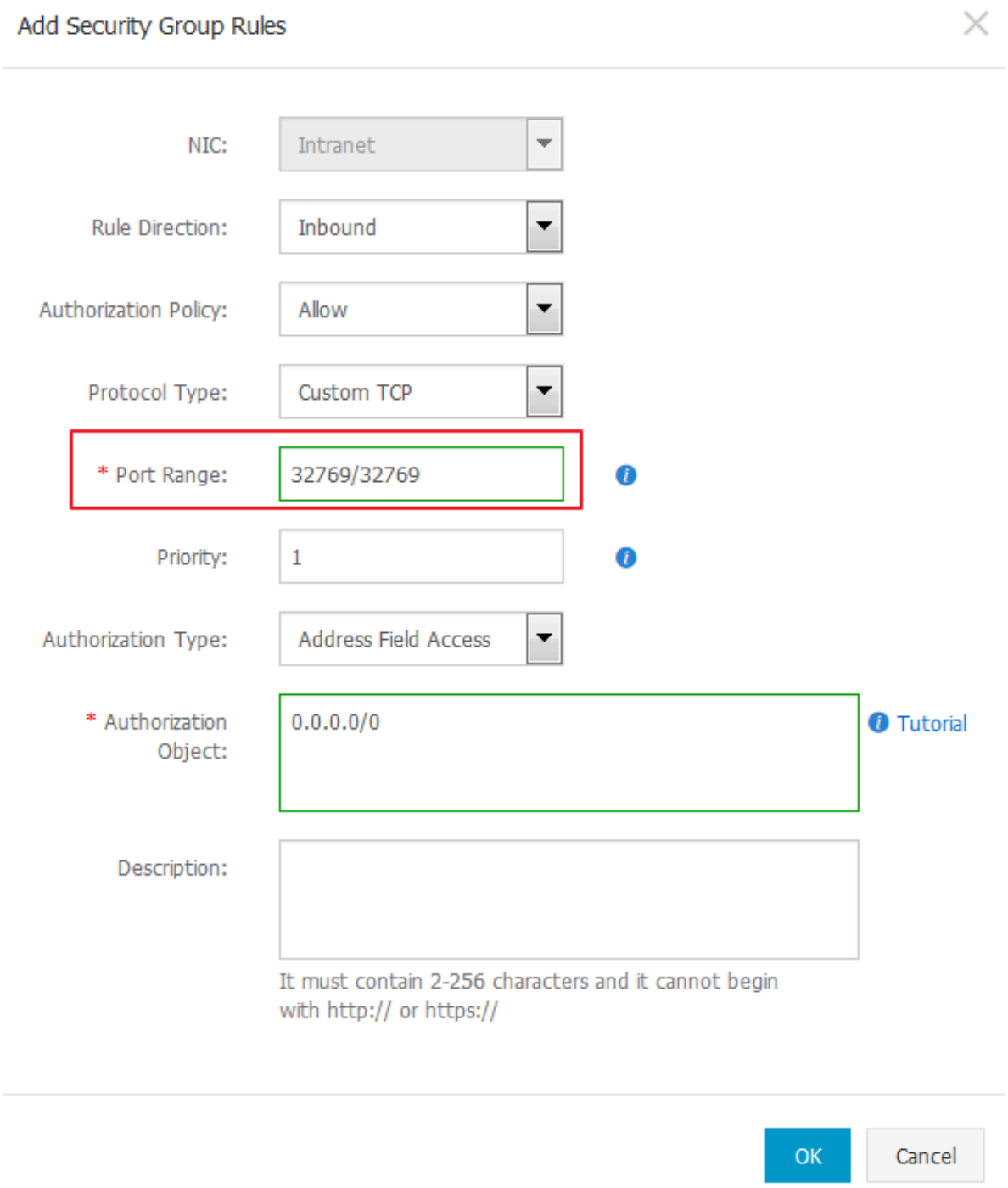


3. Container Service アプリケーション mydevbox に対応する ECS インスタンスの右側にある [詳細] をクリックします。 > ドロップダウンリストから [セキュリティグループの設定] を選択します。

4. Container Service クラスターに対応するセキュリティグループの右側にある [ルールの設定] をクリックします。



5. [セキュリティグループルールの追加] をクリックします。 [セキュリティグループルールの追加] ダイアログボックスが表示されます。 ルール情報を入力して [OK] をクリックします。



SSH を利用した Jupyter サービスへのアクセス

Linux の場合

Linux マシンを使用している場合、以下のコマンドを実行して、SSH により Jupyter サービスにアクセスします。

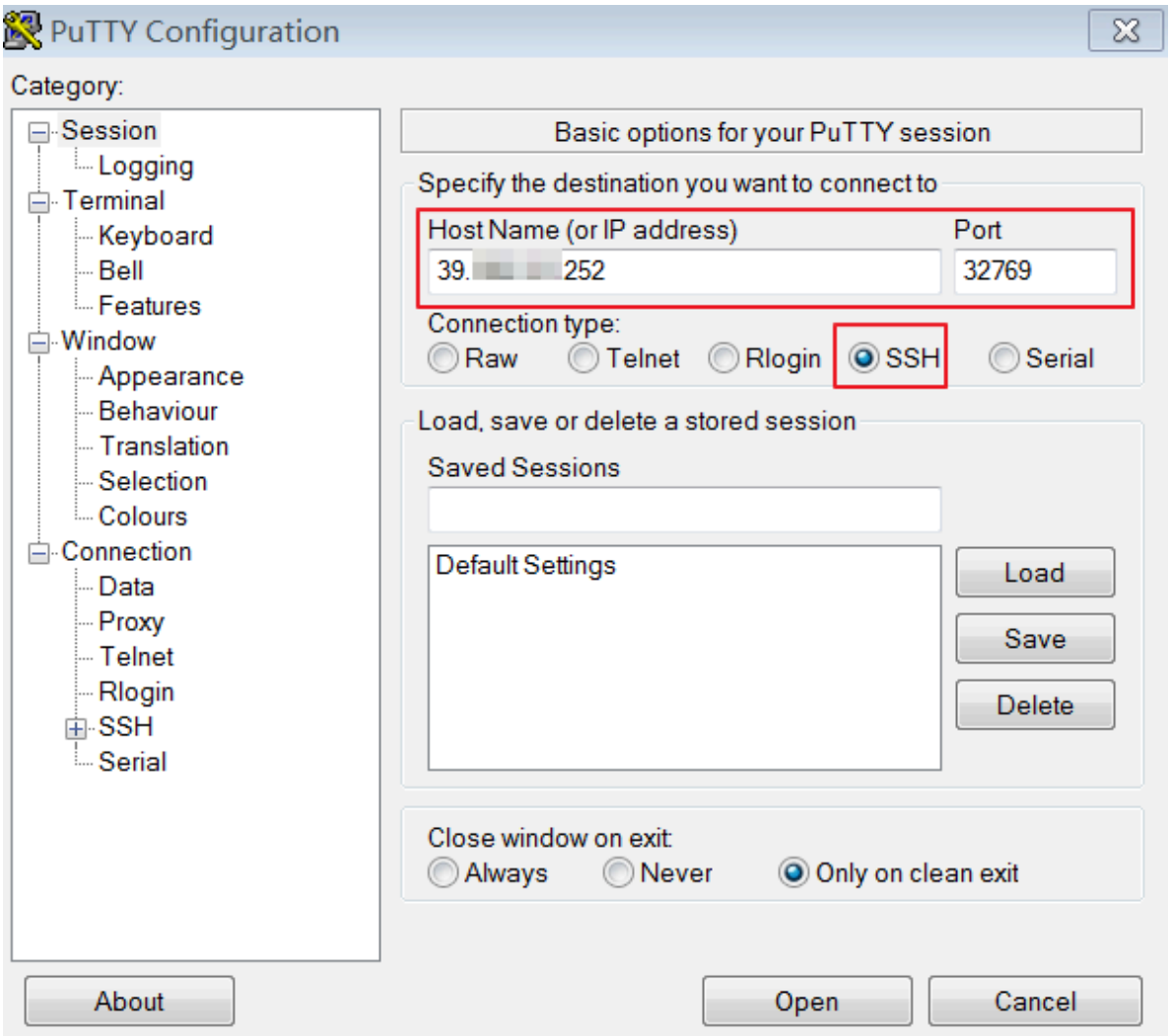
```
ssh  -p  32769  root@39.  252
```

`32769` は SSH によりアクセスするポート、`39.  252` は ECS インスタンスにバインドされた パブリック EIP です。

Windows の場合

1. PuTTY を実行して、セッションを設定します。

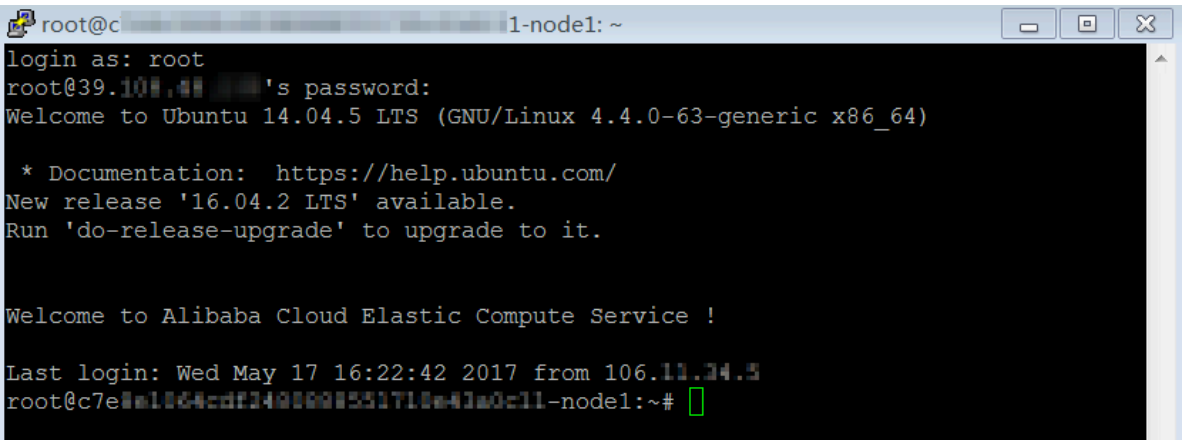
   IP アドレス (ECS インスタンスのパブリック EIP)、SSH によりアクセスするポート (このページの例では、32769) を設定し、接続タイプとして [SSH] を選択します。 次に、[開く] をクリックします。



2. 表示されたダイアログボックスで Jupyter にログインしてサービスにアクセスします。

   ログインアカウント root を入力します。 [SSH の有効化] で設定した [SSH パスワード] を入力します。 これは、*Jupyter 環境の作成*により設定されたものです。

   SSH により Jupyter サービスにアクセスできます。

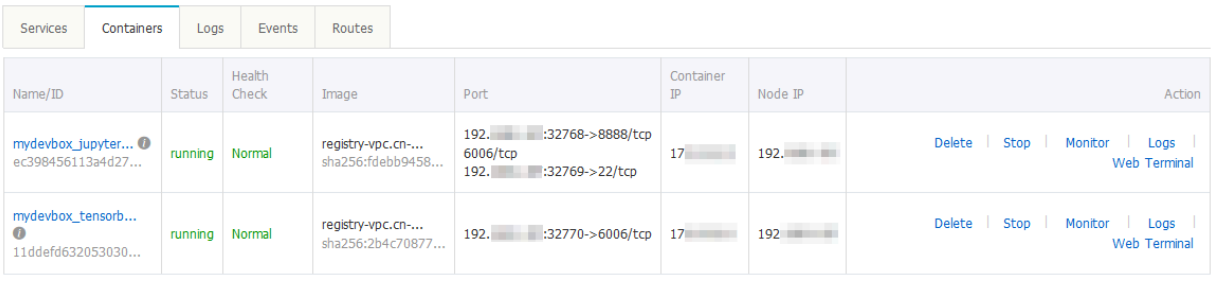


4.4 ポート転送による Jupyter サービスへのアクセス

ポート転送による Jupyter サービスへのアクセスには以下のようなメリットがあります。 ただし、設定は複雑なものとなります。

- Server Load Balancer インスタンスの購入が必要ないため、コストを抑えることができます。
- 公式 Web サイトからポートを開かず、外部から Jupyter サービスにアクセスできます。

以下の例では、Jupyter サービスはローカルポート 12345 でアクセス要求を転送することにより、SSHにアクセスします。



手順 1: SSH トンネルのセットアップ

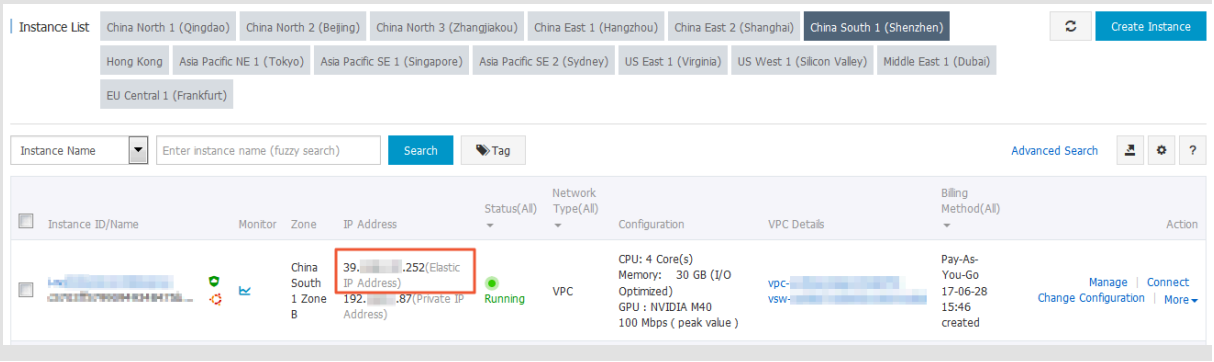
MAC OS X および Linux の場合

以下のコマンドを実行して、ローカルポートを ECS (Elastic Compute Service) インスタンスに接続します。

```
ssh -ND 12345 root@39.252
```

`12345` は使用するローカルポートです。これはカスタマイズできます。 `39.252` は、ECS インスタンスにバインドされたパブリック EIP (Elastic IP) です。

> 注：
> *ECS コンソール* にログインして ECS インスタンスにバインドされたパブリック EIP を確認できます。 ECS インスタンスにパブリック EIP がバインドされていない場合、 パブリック EIP をバインドします。 詳しくは、「*パブリック EIP アドレスの管理*」をご参照ください。



Windows の場合

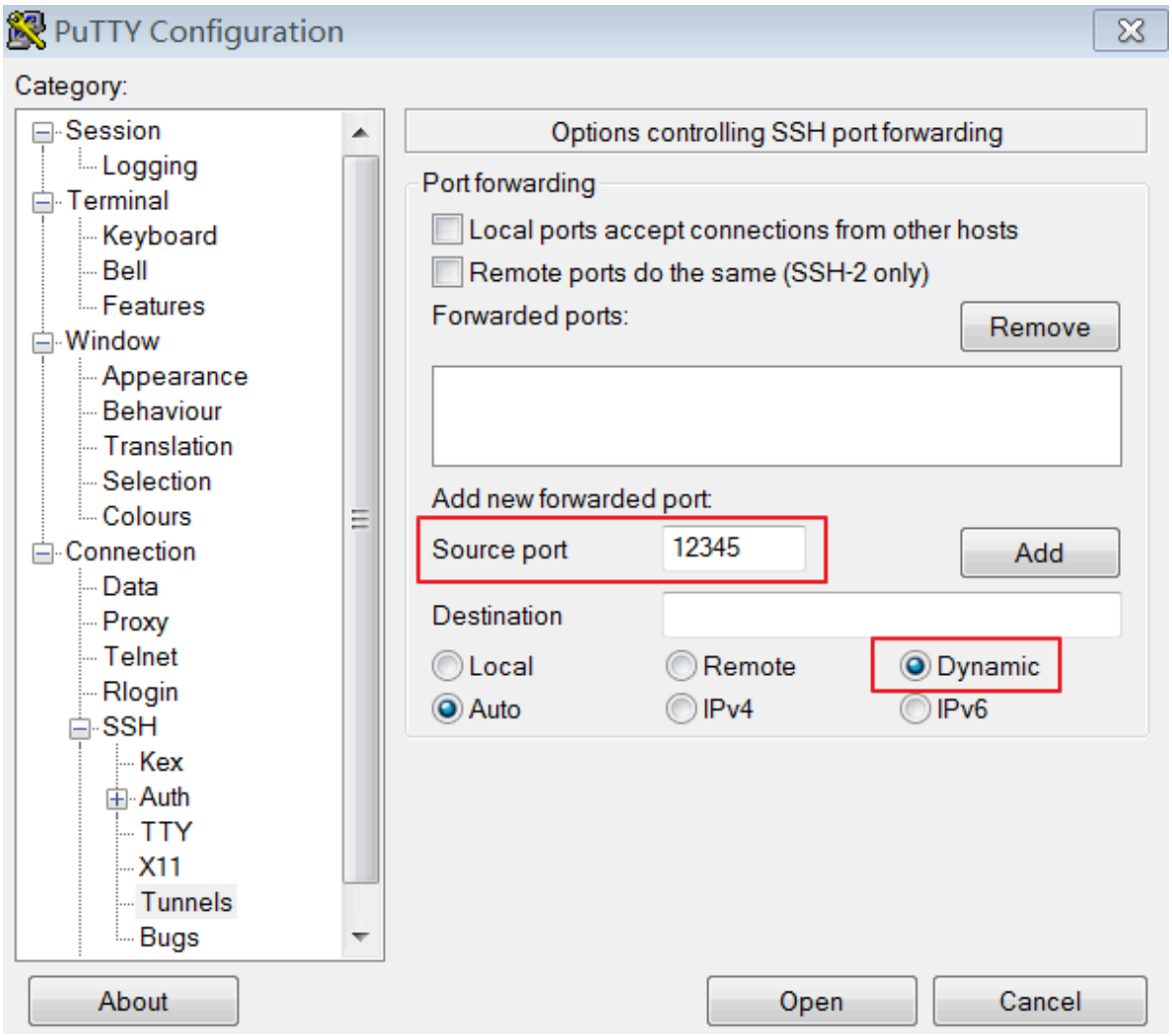
1. ローカルポート転送を設定します。

   PuTTY を実行して SSH [トンネル] を設定します。

   a. [ソースポート] を設定します。 この例では、12345 となります。

   b. [ダイナミック] を選択します。

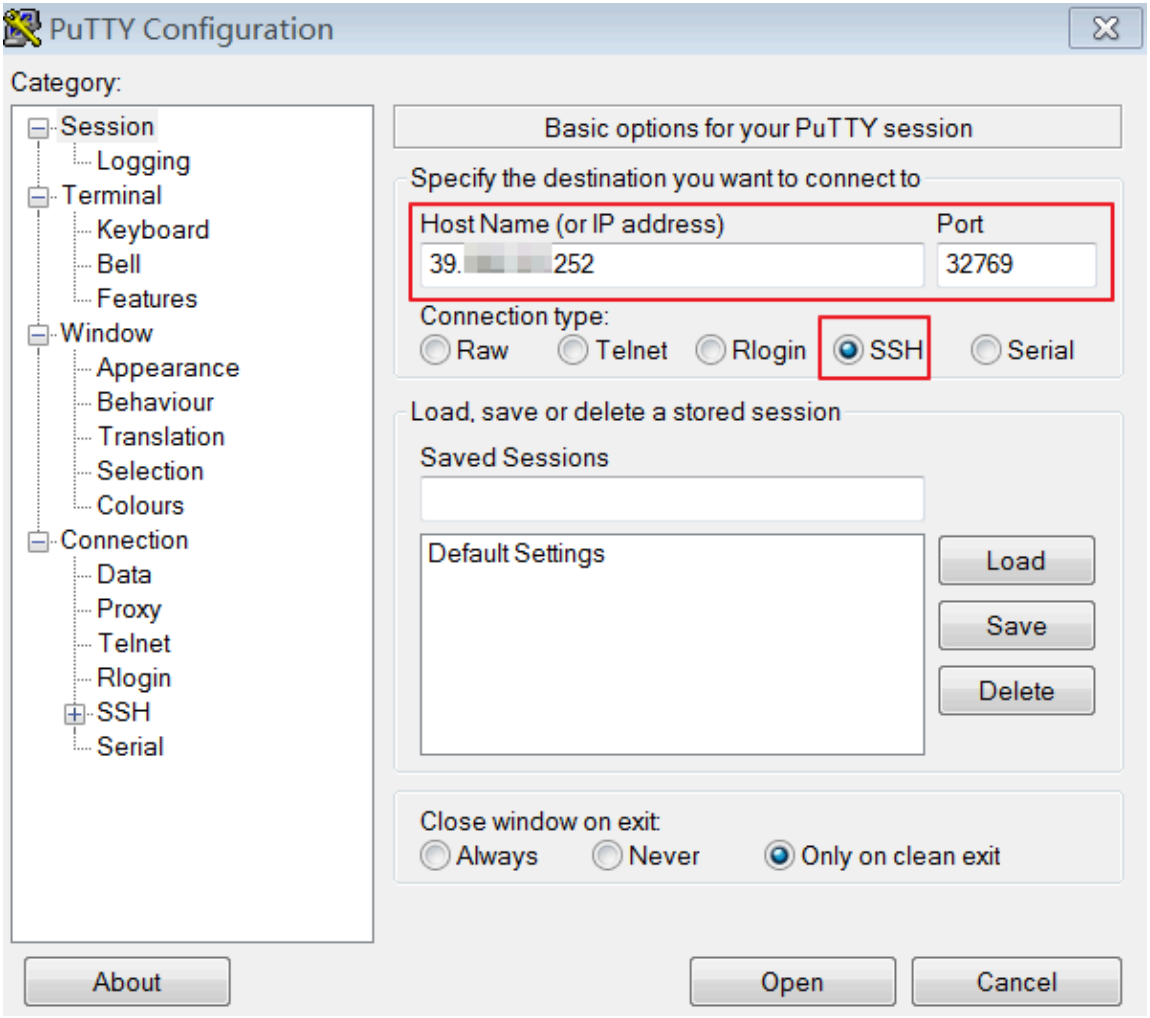
   c. [追加] をクリックします。



2. ECS インスタンスにログインします。

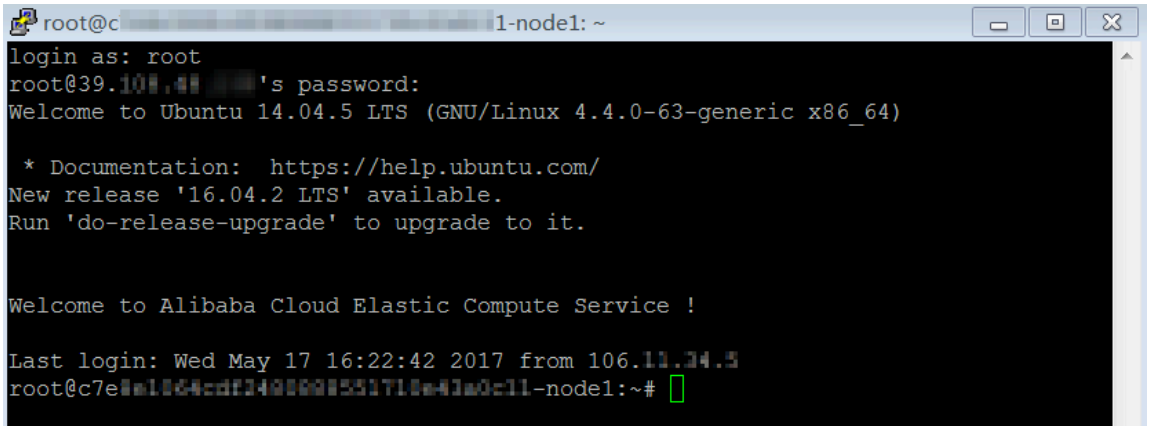
   a. PuTTY を起動します。 [セッション] を設定して、 [開く] をクリックします。

      IP アドレス、つまり ECS インスタンスのパブリック EIP を入力します。 この例では、 39 . 252 となります。



   b. 表示されたダイアログボックスで、ログインアカウントおよび ECS インスタンスのパスワードを入力します。

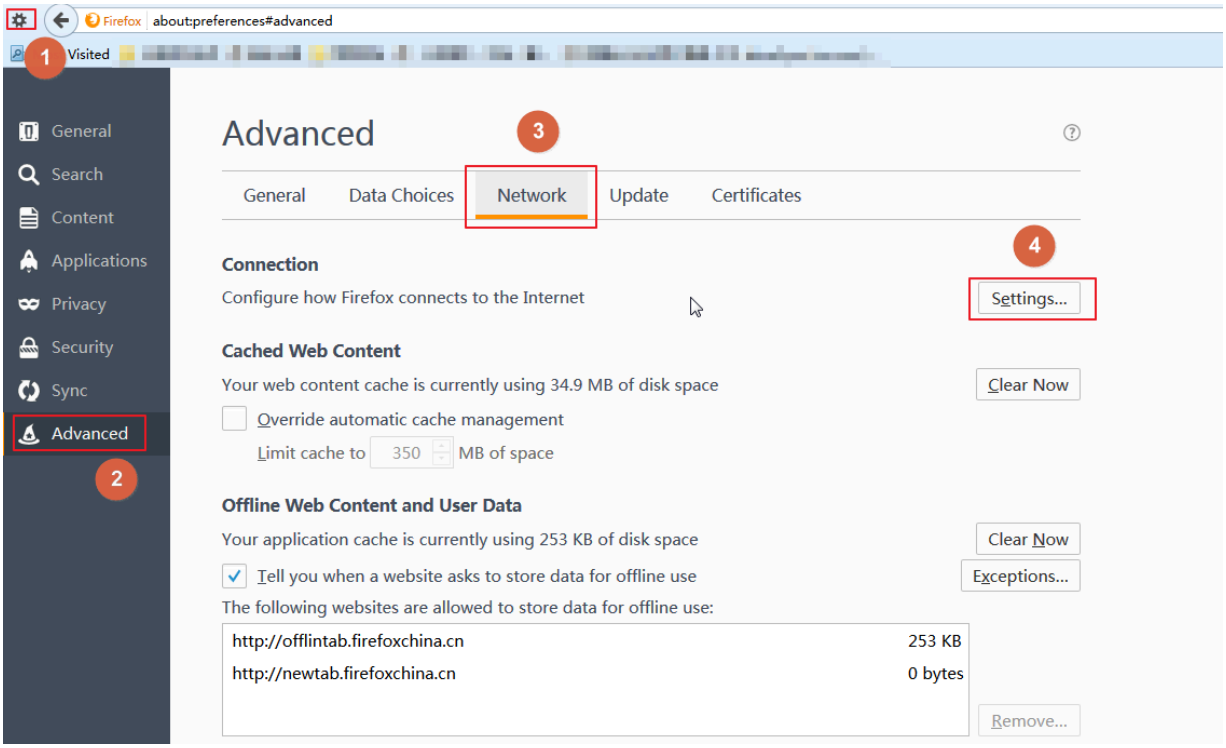
      これで、ECS インスタンスに正常にログインしました。

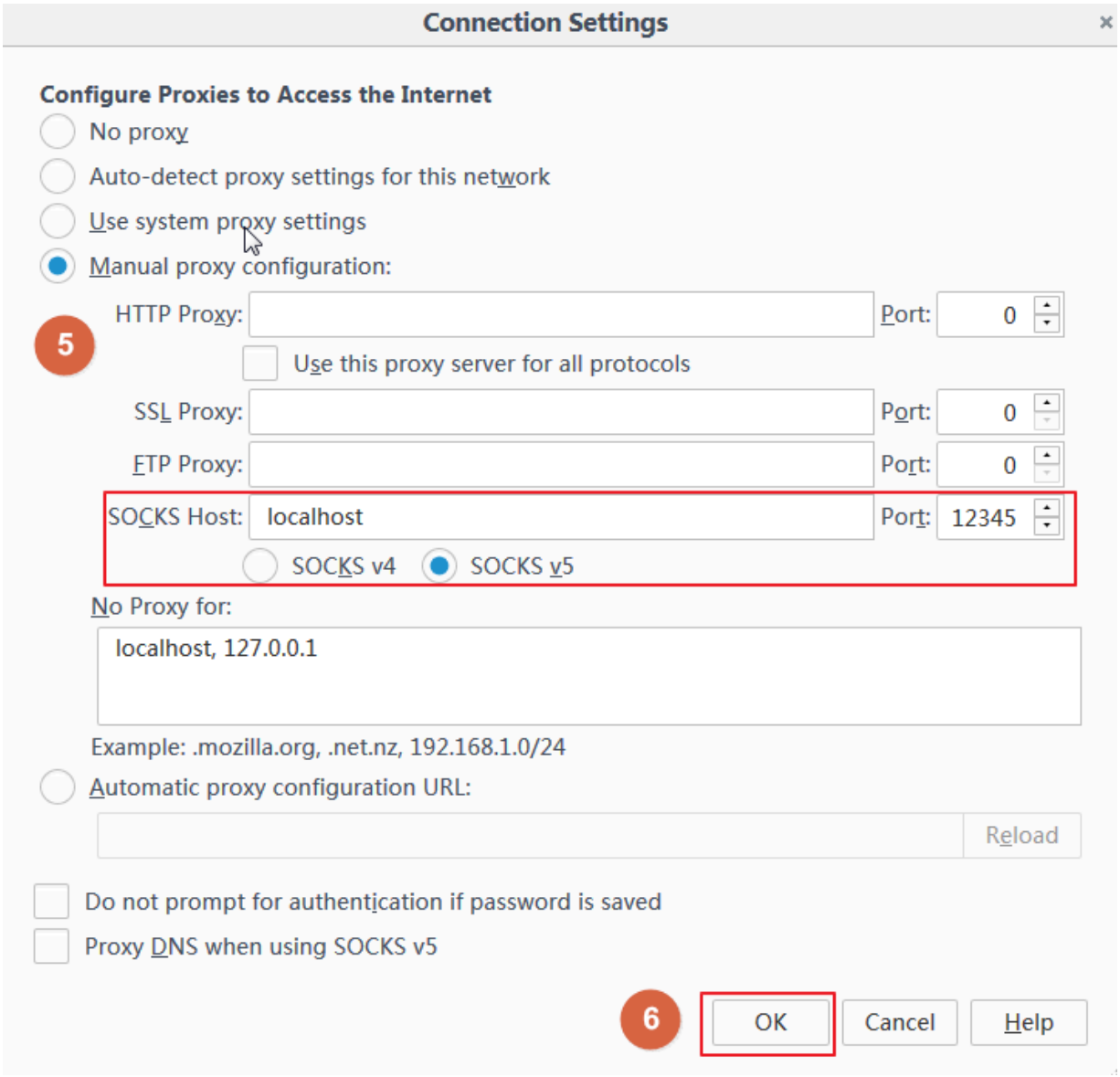


手順 2: ブラウザのネットワーク接続の設定

Firefox の場合

Firefox ブラウザを使用する場合、ブラウザを開き、[ツール] > [オプション] > [高度な設定] > [ネットワーク] をクリックして、 [接続] セクションの [設定] をクリックします。 表示されたウィンドウで、[SOCKS ホスト] を設定します。





Chrome の場合

以下のコマンドを実行します。

```
Chrome  -- proxy - server =" socks5 :// localhost : 12345 " -- host -
resolver - rules =" MAP  *  0 . 0 . 0 . 0  ,  EXCLUDE   localhost "
-- user - data - dir =/ tmppath /
```

説明

- `12345` **は使用するローカルポートです。**
- **Windows では、** `/ tmppath /` **は** `d :/ tmppath` **または同様のパスとして記載されます。 Linux または MAC OS X では、/tmppath/ は** `/ tmp /` **として記載されます。**

Chrome の場所はオペレーティングシステムによって異なり、以下の表のようになります。

オペレーティングシステム	Chrome の場所
Mac OS X	/ Applicatio ns / Google \ Chromapp / Contents / MacOS / Google \ Chrome
Linux	/ usr / bin / google - chrome
Windows	C :\ Program Files (x86)\ Google \ Chrome \ Applicatio n \ chrome . exe

Windows を例にとります。

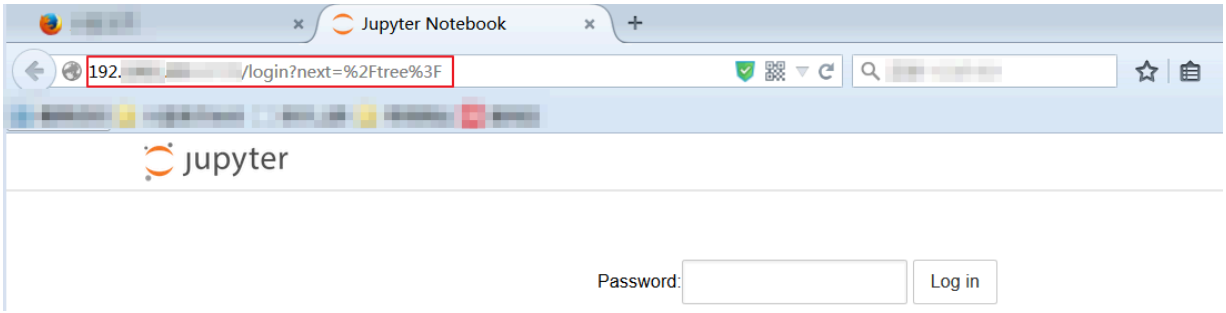
```
C:\Users >"C:\Program Files (x86)\Google\Chrome\Application\chrome.ex
e"  --proxy-server="socks5://localhost:12345" --host-resolver-rules="MAP * 0.0.0
.0 , EXCLUDE localhost" --user-data-dir=d:/tmppath

C:\Users\ >
```

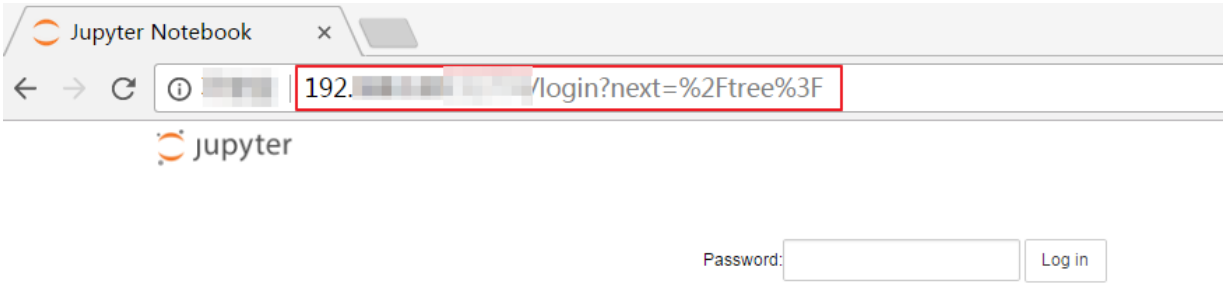
手順 3: Jupyter サービスへのアクセス

お使いのブラウザで、Jupyter サービスのアクセスアドレスを入力します。 このページの例では、アドレス 192. 83:32769 にアクセスします。 これにより、外部から SSH により Jupyter サービスにアクセスできます。

Firefox の場合



Chrome の場合



5 モデルトレーニング

5.1 スタンドアロンモデルトレーニング

Alibaba Cloud により提供されるエラスティックコンピューティングリソースおよびストレージサービスを使用してモデルトレーニングコードを実行すると、スタンドアロントレーニングの繰り返しを素早く行うことができます。 トレーニング中、いつでもログの確認およびトレーニングステータスのモニタリングが可能です。

前提条件

モデルトレーニングタスクの実行前に、以下の操作を実行したことを確認します。

- 一定数のエラスティックコンピューティングリソース (ECS (Elastic Compute Service) または EGS) を含むコンテナークラスターを作成します。 詳しくは、コンテナークラスターの作成をご参照ください。
- モデルトレーニングに関するデータの保存に OSS (Object Storage Service) を使用するため、同一アカウントを使用して OSS バケットを作成します。 次に、前述のコンテナークラスター上にデータボリュームを作成して、 OSS バケットをローカルディレクトリとしてトレーニングタスクを実行するコンテナーにマウントします。 詳しくは、データボリュームの作成をご参照ください。
- モデルトレーニングコードを利用可能な GitHub コードリポジトリに同期させます。

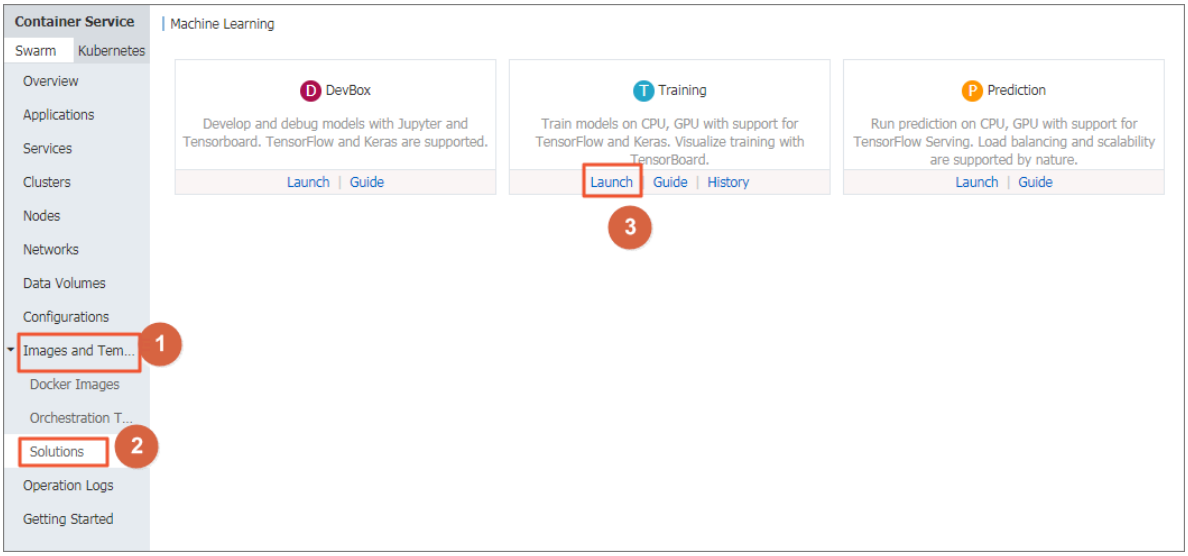
規則

トレーニングコードでトレーニングデータの読み込み、トレーニングログの出力およびトレーニングの反復ステータスデータ (チェックポイント) の保存をするために、以下の規則に注意します。

- トレーニングデータは自動的に `/input` ディレクトリに保存されます。 これは OSS 上のパスと一致します。 コードはこのディレクトリからトレーニングデータを読み込みます。
- ログおよびチェックポイントファイルを含む、コードによって出力されるすべてのデータは、`/output` ディレクトリに 格納されます。 `/output` ディレクトリに格納されるすべてのファイルは、同じディレクトリ構造を持つ OSS バケットと自動的に同期されます。
- 特別な Python 依存リポジトリがトレーニングコードに必要な場合、すべての依存関係を `requirements.txt` 設定ファイルに書き込み、GitHub リポジトリのコードのルートディレクトリに格納します。

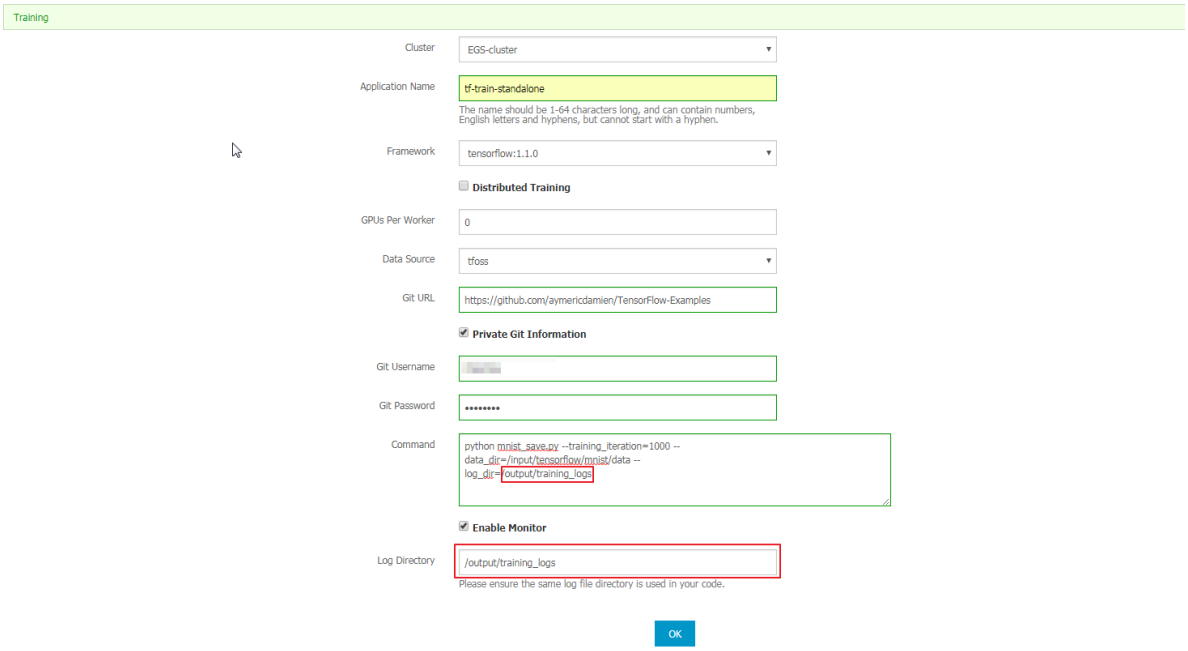
手順

1. *Container Service コンソール*にログインします。
2. 左側のナビゲーションウィンドウから [イメージとテンプレート] > > [ソリューション]をクリックします。
3. [トレーニング] の [起動] をクリックします。



4. スタンドアロントレーニングタスクに関する基本情報を設定します。

   以下の設定を完了させます。



   - クラスター: スタンドアロンモデルトレーニングタスクを実行するコンテナークラスターを選択します。
   - アプリケーション名: スタンドアロンモデルトレーニングタスクを実行するアプリケーションの名前で、 1 文字以上 64 文字以内で、数字、英字およびハイフン (-) を含むことができます。 たとえば、"tf-train-standalone" となります。
   - フレームワーク: モデルトレーニングに使用するフレームワークを選択します。 現在、サポートされるフレームワークには、TensorFlow、 Keras、Python およびカスタマイズイメージが含まれます。 カスタマイズイメージを選択した場合、[イメージ] フィールドに有効なイメージのアドレスを入力します。
   - 分散トレーニング: 分散トレーニングタスクを実行するかどうかを選択します。 ここでは、スタンドアロントレーニングを実行するため、チェックボックスをオフにします。
   - ワーカーごとの GPU: 使用する GPU 数を指定します。 このパラメーターを "0" に設定すると、 GPU の代わりに CPU を使用してトレーニングを行います。
   - データソース: トレーニングデータの保存に使用するデータソースを選択します。 OSS によりクラスター上に作成されたデータボリュームを選択するか、ローカルディレクトリを

選択して絶対パスを入力 入力します。 [データソースなし] を選択ですることもできます。この例では、tfoss データボリュームを選択します。

- Git URL: トレーニングコードが保存されている GitHub コードリポジトリのアドレスを指定します。

  > 注 :
  > 現在、HTTP および HTTPS がサポートされますが、SSH はサポートされていません。

- プライベート Git 情報: プライベートコードリポジトリを使用する場合は、このチェックボックスをオンにします。 次に、 Git ユーザー名 (お使いの GitHub アカウント) および Git パスワード を設定します。
- コマンド: モデルトレーニングを行う上記のコードの実行に使用するコマンドを指定します。

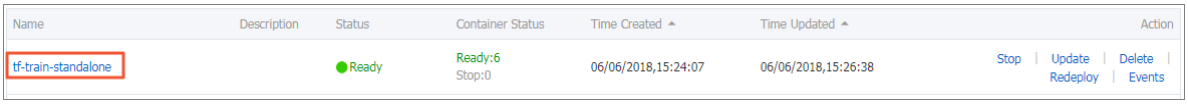
  > 注 :
  > Python3 をサポートするフレームワークを選択した場合、 `python` の代わりに直接 `python3` をコマンドラインで呼び出します。

- モニターの有効化: TensorBoard を使用してトレーニングデータをモニタリングするかどうかを 選択します。 このチェックボックスをオンにした場合、[ログディレクトリ] フィールドにトレーニングデータを保存する有効なパスを入力して、パスがトレーニングコードでのログ出力パスと同じものであることを確認します。 たとえば、 [ログディレクトリ] を *`/ output / training_l ogs`* に設定した場合、コードはログを同じパスに出力します。 上の図に示すように、 コードのコマンドラインパラメーター `-- log_dir` の指定によりパスの整合性が保証されます。

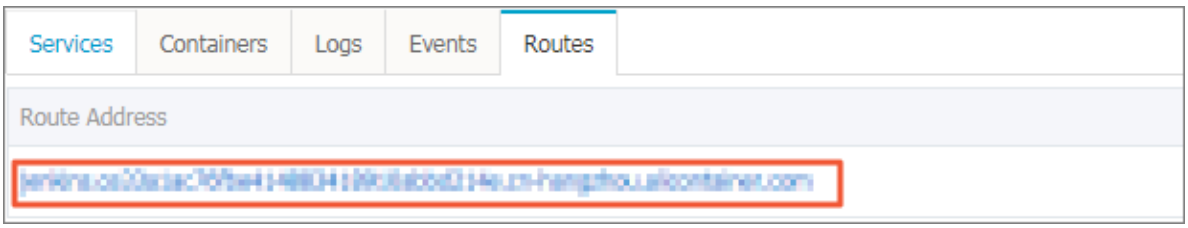
  > 注 :
  > ここで、トレーニングログとは TensorBoard 用の TensorFlow API により出力されたイベントファイル、 およびモデルステータスを保存するチェックポイントファイルを示します。

5. 設定完了後、[OK] をクリックして、モデルトレーニングタスクの実行に使用するアプリケーションを作成します。

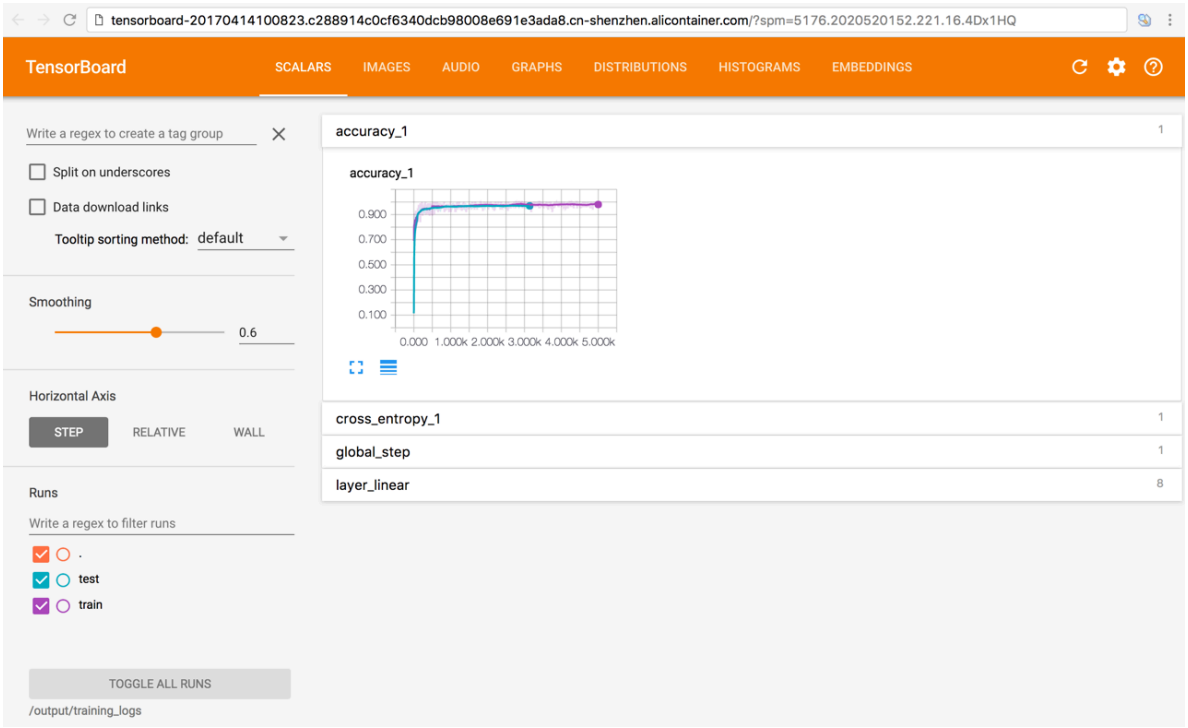
6. 左側のナビゲーションウィンドウから [アプリケーション] をクリックします。 [クラスター] ドロップダウンリストからクラスターを選択してから、作成したアプリケーションの名前 (この例では、tf-train-standalone) をクリックします。



7. [ルート] タブをクリックします。 "tensorboard" から始まるリンクが表示されます。



このリンクをクリックして TensorBoard モニタリングページを開きます。 トレーニングモニタリングデータを参照できます。



8. [ログ] タブをクリックすると、トレーニングタスクが実行中のときに、コードによってstdout/stderr に出力されるログが表示されます。

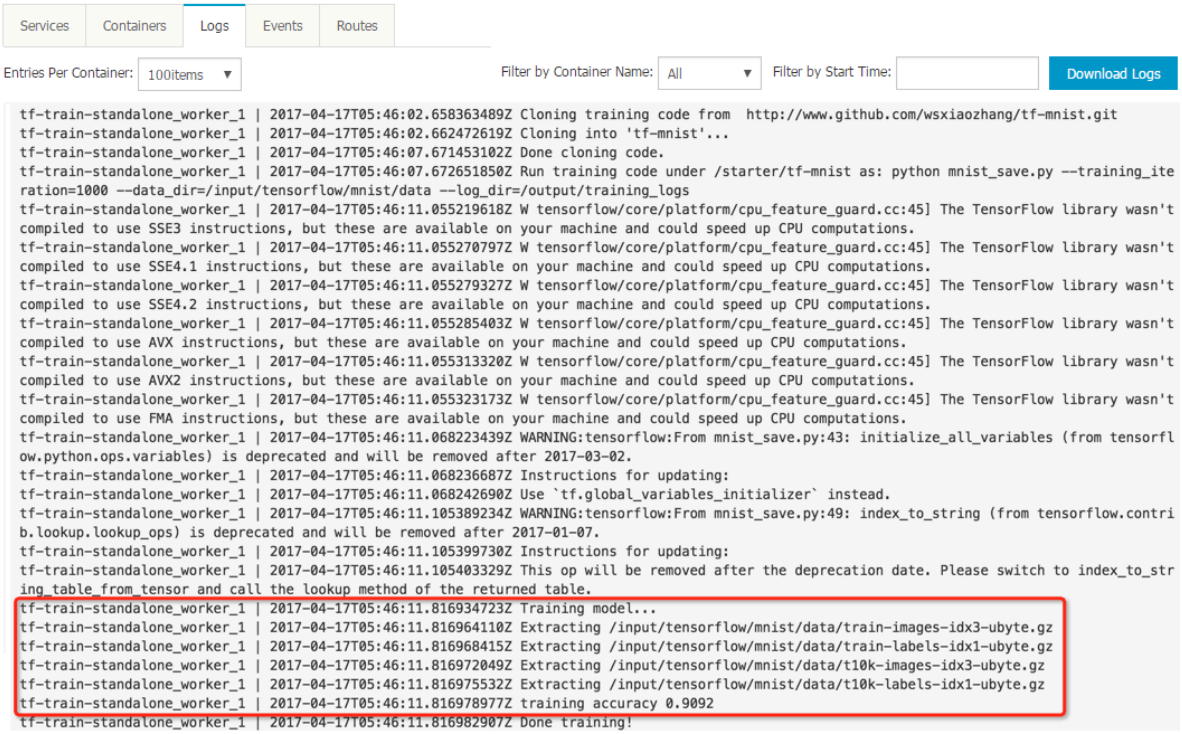
注：

一般的に、トレーニングタスクアプリケーションの実行のために、複数のサービスコンテナーが自動的に作成され、それぞれ異なるプログラムブランチを実行します。 たとえば、ワーカーコンテナーは通常モデルコードの実行に使用されますが、tensorboard コンテナー

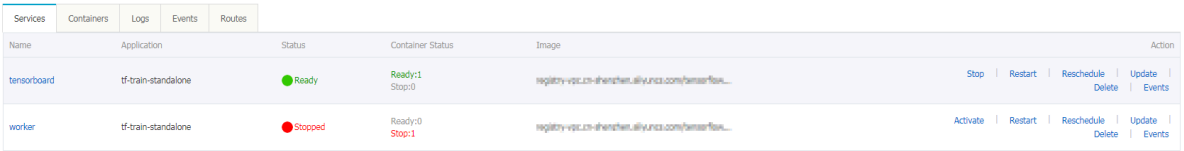
は TensorBoard トレーニングモニタリングの実行に使用されます。 [サービス] タブまたは [コンテナー] タブをクリックして作成された サービスまたはコンテナーを表示できます。

たとえば、 `tf - train - standalone _worker_1` のようにサービスコンテナー名でログをフィルタリングでき、コンテナー操作プログラムにより stdout/stderr に出力された詳細なログを参照できます。 時間および表示量によりログをフィルタリングすることもでき、ログファイルのローカルディレクトリへのダウンロードも選択できます。

注：
ここでの stdout/stderr ログは、上記の TensorBoard イベントファイルおよびチェックポイントファイルとは異なります。



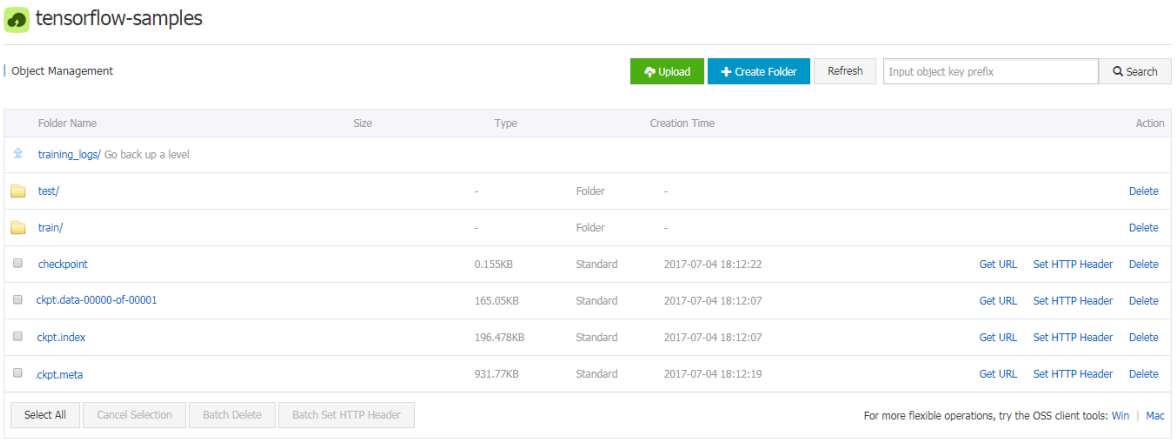
9. [サービス] タブをクリックします。 ワーカーサービスのステータスが [停止] の場合、トレーニングタスクが完了しています。



10. OSS クライアントで、OSS バケットに自動的に格納されたトレーニング結果を確認できます。

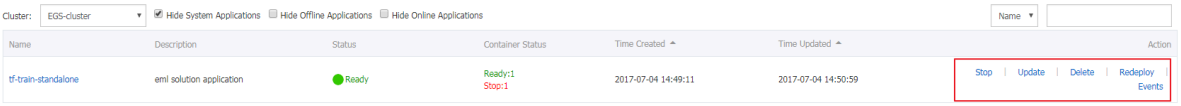
トレーニング完了後、ローカルの `/output` ディレクトリに格納されたすべてのファイルが、システムにより自動的に指定したデータボリュームに対応する OSS バケットにコピーさ

れます。 OSS クライアントでは、 `/ output` ディレクトリに書き込まれたイベントファイルおよびチェックポイントファイルが OSS バケットにバックアップされたことを 確認できます。



11.トレーニングタスクを管理します。

トレーニングタスクの停止、再起動または削除するには (たとえば、新しいトレーニングタスクのためにコンピューティングリソースをリリースするため)、左側のナビゲーションウィンドウから [アプリケーション] をクリックして アプリケーションリストページに戻ります。対応するアプリケーションを検索して、右側にある操作を実行します。



注：

ここで説明しているモデルトレーニングサービスを使用することで、最初からモデルをトレーニングすることができるだけでなく、新しいデータを使用して既存のモデル (チェックポイント) を 基にしたトレーニング (たとえば、微調整) を継続することもできます。 既存のアプリケーションの設定を継続的に更新することで、ハイパーパラメーターを調整し、反復的なトレーニングを実行できます。

5.2 分散モデルトレーニング

Alibaba Cloud により提供されるエラスティックコンピューティングリソースおよびストレージサービスを使用して分散トレーニングを迅速に行うことで、モデルトレーニングコードを実行できます。 トレーニング中、コンピューティングリソース (CPU および GPU) の分配方法の制御、ログの確認、およびトレーニングステータスのモニタリングをいつでも行え、トレーニング結果をストレージサービスにバックアップできます。

このドキュメントで説明するモデルトレーニングサービスを使用することで、最初からモデルをトレーニングするだけでなく、新しいデータを使用して既存のモデル (チェックポイント) に基づいたトレーニングの継続 (たとえば、微調整) も 行えます。 既存のアプリケーションの設定を継続的に更新することで、ハイパーパラメータを調整し、反復的なトレーニングを実行できます。

前提条件

モデルトレーニングタスクの実行前に、以下の操作を実行したことを確認します。

- 一定数の エラスティックコンピューティングリソース (ECS (Elastic Compute Service) または EGS) を含むコンテナークラスターを作成します。 詳しくは、コンテナークラスターの作成をご参照ください。
- モデルトレーニング用データの保存に OSS (Object Storage Service) を使用するために、同じアカウントで OSS バケットを作成します。 次に、前述のコンテナークラスターでデータボリュームを作成し、 トレーニングタスクを実行するコンテナーにローカルディレクトリとして OSS バケットをマウントします。 詳しくは、データボリュームの作成をご参照ください。
- モデルトレーニングコードを利用可能な GitHub コードリポジトリと同期させます。

規則

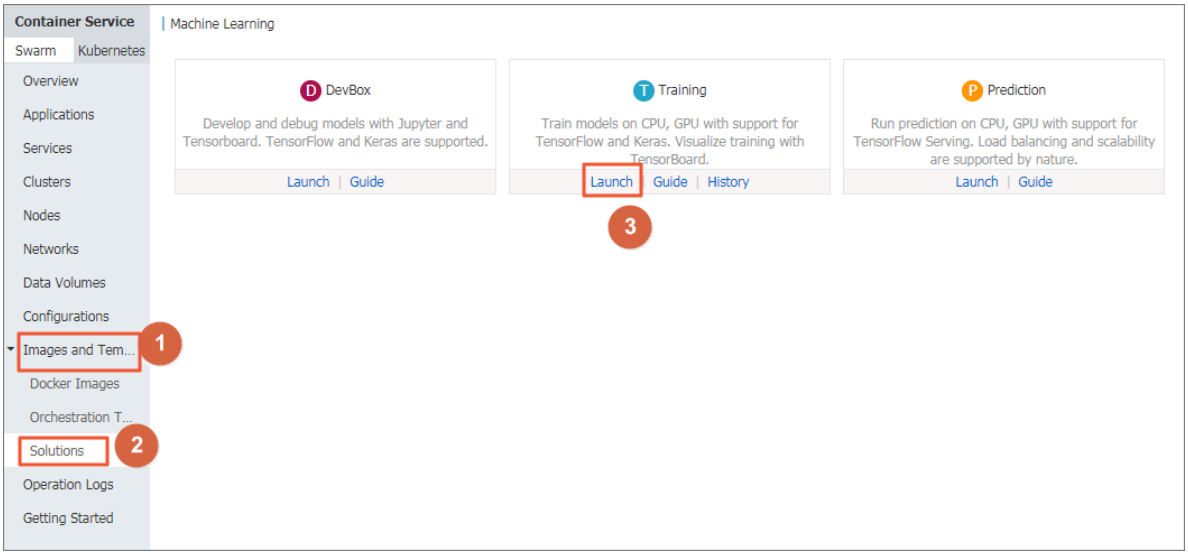
トレーニングコードでトレーニングデータを読み込み、トレーニングログを出力し、トレーニングの反復ステータスデータ (チェックポイント) の保存を簡単にするために、以下の規則にご注意ください。

- トレーニングデータは自動的に `/input` ディレクトリに保存されます。 これは OSS 上のパスと一致します。 コードはこのディレクトリからトレーニングデータを読み込みます。
- ログおよびチェックポイントファイルを含む、コードによって出力されるすべてのデータは、`/output` ディレクトリに 保存されます。 `/output` ディレクトリに保存されるすべてのファイルは、同じディレトリ構造を持つ OSS バケットに 自動的に同期されます。特別な Python 依存リポジトリがトレーニングコードに必要な場合、すべての依存関係を `requirements.txt` 設定ファイルに書き込み、そのファイルを GitHub リポジトリのコードルートディレクトリに格納します。

手順

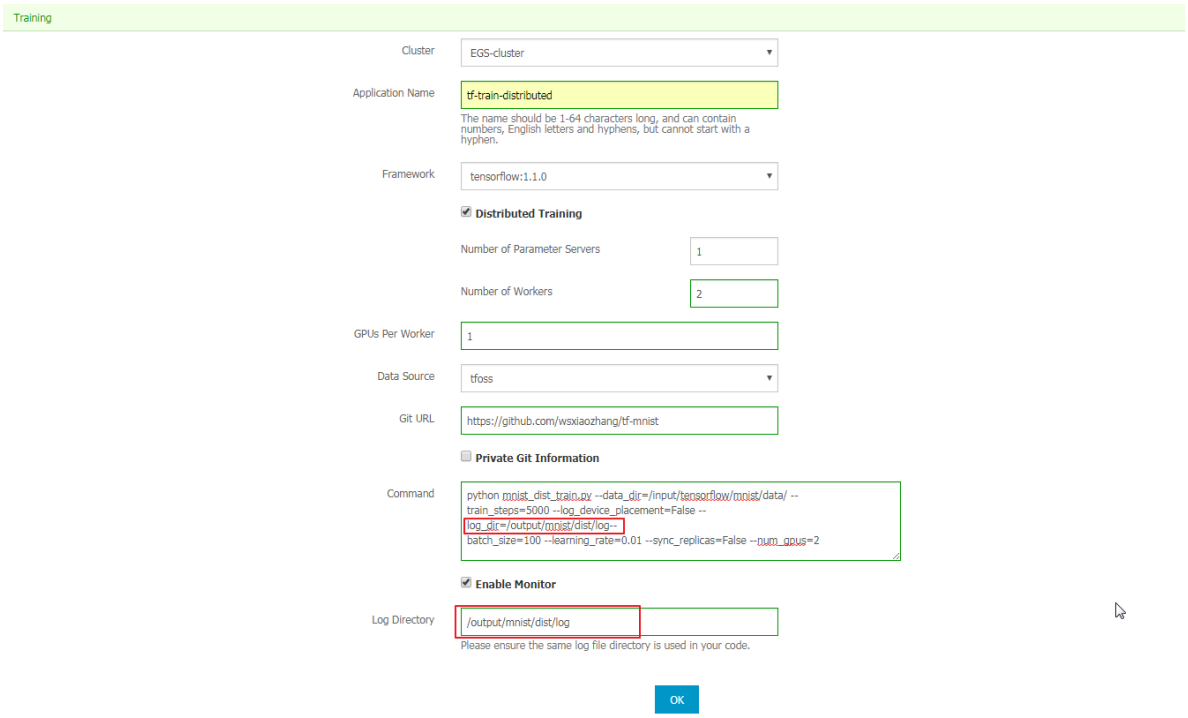
1. *Container Service コンソール*にログインします。
2. Swarm で、左側のナビゲーションウィンドウから [イメージとテンプレート] > > [ソリューション] をクリックします。

3. [トレーニング] の [起動] をクリックします。



4. 分散トレーニングタスクに関する基本情報を設定します。

   以下の設定を完了させます。



   - クラスター: 分散モデルトレーニングタスクを実行するコンテナークラスターを選択します。
   - アプリケーション名: トレーニングタスクの実行用に作成するアプリケーションの名前は、1 文字以上 64 文字以内で、数字、英字およびハイフン (-) を含むことができます。 たとえば、"tf-train-distributed" となります。
   - フレームワーク: モデルトレーニングに使用するフレームワークを選択します。 現在、サポートされるフレームワークには、 TensorFlow、Keras、Python およびカスタマイズイメージが含まれます。 カスタマイズイメージを選択した場合、[イメージ] フィールドに有効なイメージのアドレスを入力します。
   - 分散トレーニング: 分散トレーニングを実行するかどうかを選択します。 ここでは、分散トレーニングを実行するため、チェックボックスをオンにします。 パラメーターサーバー数およびワーカー数を設定し、 分散トレーニング中の計算に使用するパラメーターサーバーノード数および ワーカーノード数を指定します。 この例では、1 つのパラメーターサーバーと 2 つのワーカーを設定します。
   - ワーカーごとの GPU: トレーニングタスクが実行される際のそれぞれのワーカーに対して使用する GPU 数を指定します。 このパラメーターを "0" に設定すると、GPU の代わり

に CPU を使用してトレーニングを行います。 この例では、それぞれのワーカーで 1 つの GPU を使用します。

- データソース: トレーニングデータの保存に使用するデータソースを選択します。 クラスターで OSS により作成されたデータボリュームまたはローカルディレクトリを選択し、絶対パスを 入力します。 [データソースなし] を選択でするこもできます。 この例では、"tfoss" データボリュームが選択されてます。
- Git URL: トレーニングコードが格納されている GitHub コードリポジトリのアドレスを指定します。

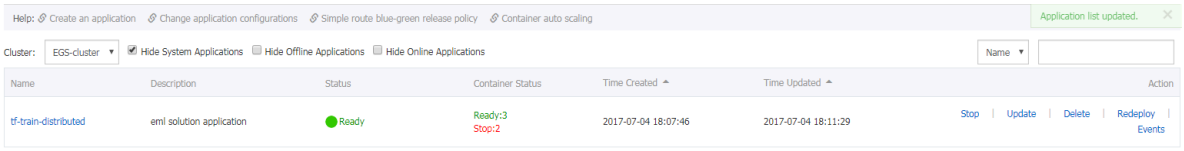
  
  注 :
  現在、HTTP および HTTPS がサポートされますが、SSH はサポートされていません。
- プライベート Git 情報: プライベートコードリポジトリを使用する場合は、このチェックボックスをオンにします。 次に、Git ユーザー名 (お使いの GitHub アカウント) および Git パスワードを設定します。
- コマンド: モデルトレーニングを行う上記のコードの実行に使用するコマンドを指定します。

  
  注 :
  Python3 をサポートするフレームワークを選択した場合、 `python` の代わりに直接 `python3` をコマンドラインで呼び出します。
- モニターを有効にする: トレーニングステータスをモニターするために TensorBoard を使用するかどうかを 選択します。 このチェックボックスにチェックを入れて、[ログディレクトリ] フィールドにトレーニングデータを格納する有効なパスを入力して、そのパスがトレーニングコードのログ出力パスと同じものであることを確認します。 たとえば、 [ログディレクトリ] を `/ output / training_l ogs` に設定した場合、コードはログを同じパスに出力します。 上の図に示すように、 パスの整合性はコードのコマンドラインパラメーター `-- log_dir` を指定することで保証されます。

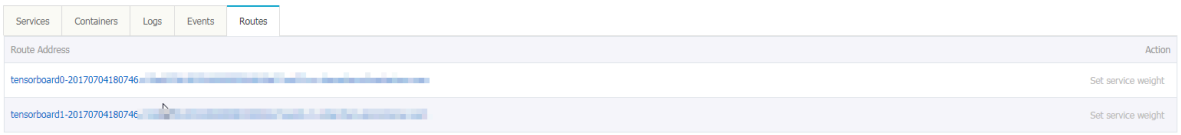
  
  注 :
  ここで、トレーニングログとは TensorBoard 用の TensorFlow API により出力されたイベントファイル、 およびモデルの状態を格納するチェックポイントファイルを示しています。 (TensorFlow を利用した分散トレーニングの実行時、主要なロールを行うワーカーが、通常チェックポイントを保存するロールを担います)。

5. 設定完了後、[OK] をクリックして、モデルトレーニングタスクの実行に使用するアプリケーションを作成します。

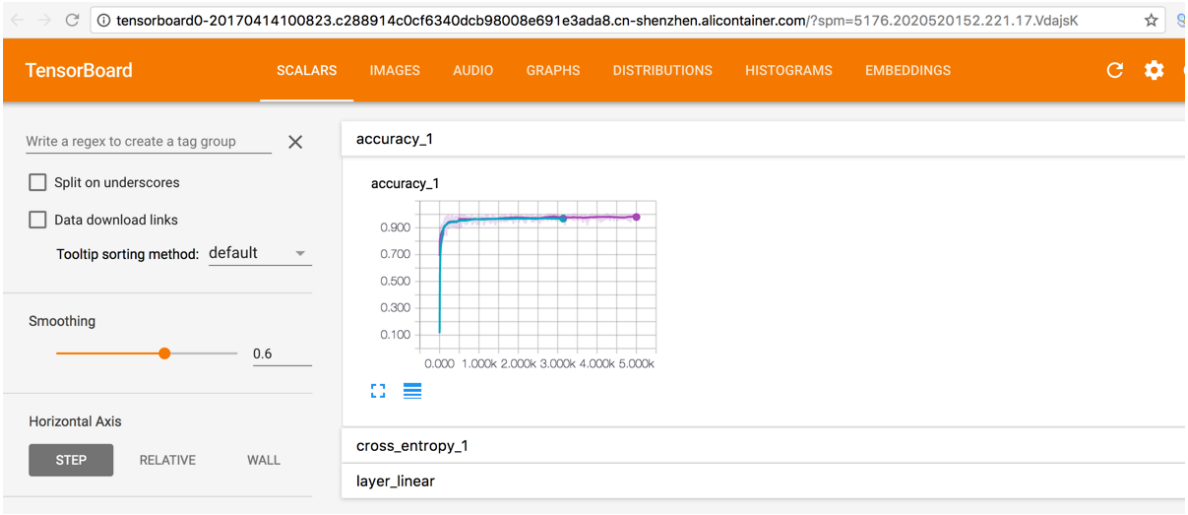
6. 左側のナビゲーションウィンドウから [アプリケーション] をクリックします。 [クラスターリスト] からクラスターを選択して、作成したアプリケーションの名前をクリックします (この例では tf-train-distributed となります)。



7. [ルート] タブをクリックします。 "tensorboard" で始まる 2 つのリンクが表示されます。 それぞれのリンクを使用して対応するワーカーノードの TensorBoard モニタリングページを開きます。 "tensorboard0" で始まるリンクは ワーカー "0" を示し、"tensorboard1" で始まるリンクは ワーカー "1" を示します。



リンクをクリックしてトレーニングモニタリングデータを表示します。



8. [ログ] タブをクリックして、トレーニングタスクの実行時にお使いのコードにより stdout/stderr に出力されたログを表示します。

> **注：**
> 通常、トレーニングタスクアプリケーションの実行のために、複数のサービスコンテナーが自動的に作成され、それぞれ異なるプログラムブランチを実行します。 たとえば、"ps" コンテナーはパラメーターサーバーコードの実行、 "woker" コンテナーはモデルコンピューティングコードの実行、"tensorboard" コンテナーは TensorBoard トレーニングモニタリング

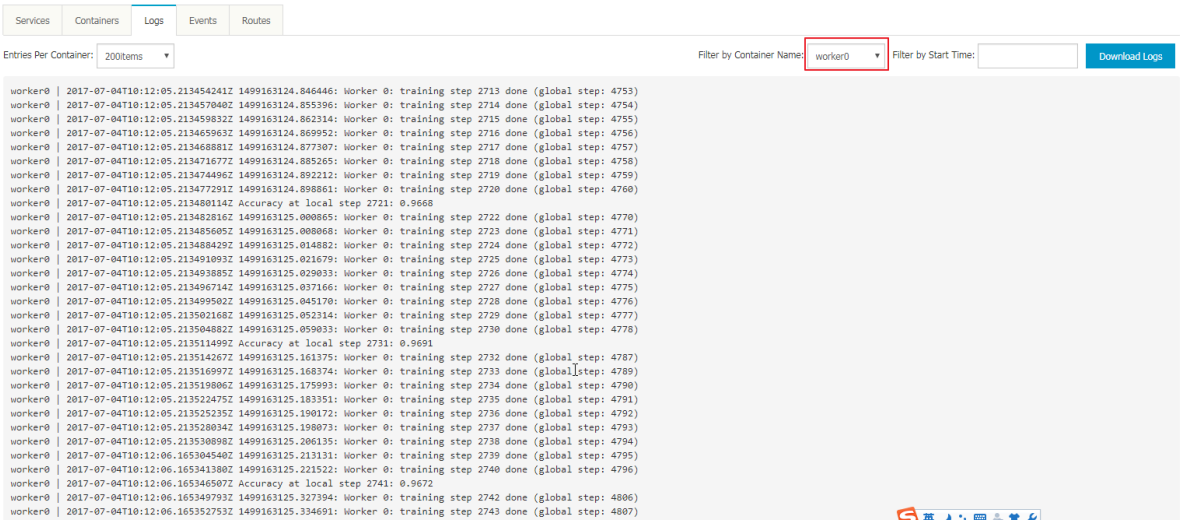
の実行に使用されます。 [サービス] タブまたは [コンテナー] タブをクリックして作成されたサービスまたはコンテナーを 表示できます。

たとえば、"worker0" のようにサービスコンテナー名でログをフィルタリングでき、 コンテナー操作プログラムにより stdout/stderr に出力された詳細なログを参照できます。 時間と表示量によりログをフィルタリングすることもでき、ログファイルのローカルディレクトリへのダウンロードも選択できます。 この例では、"worker0" および "worker1" のログの確認後、"worker0" と "worker1" が一部のトレーニングデータに基づいた並列モデルにより同じモデルを計算し、"ps0" を利用して同時に勾配の更新を行なっていることを確認できます。

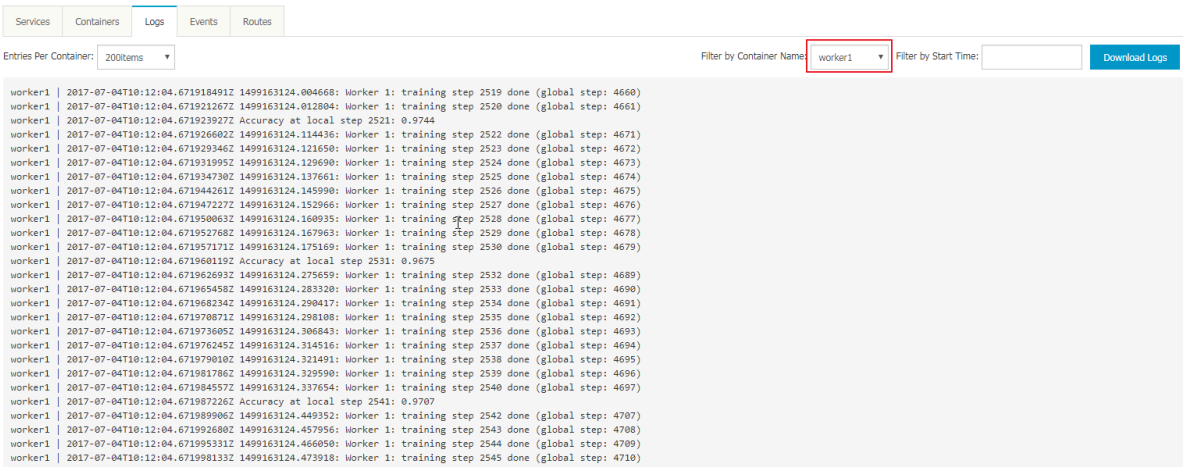
注：

ここで説明した stdout/stderr ログは、 前述の TensorBoard イベントファイルおよびチェックポイントファイルとは異なります。

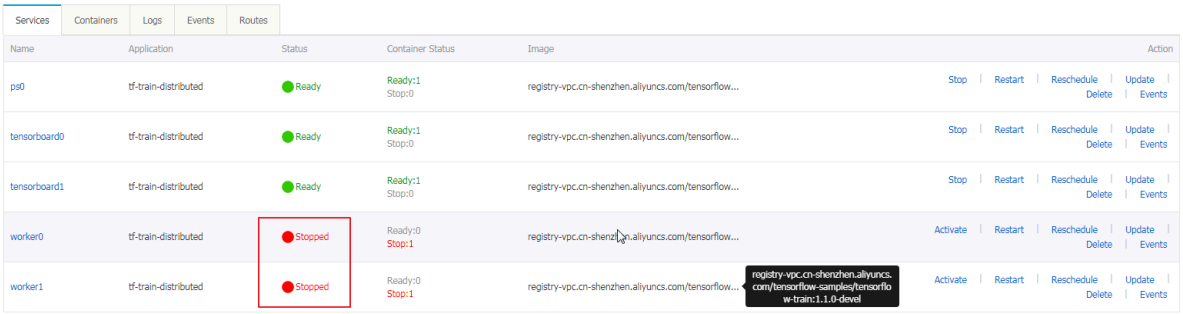
"worker0" ログ



"worker1" ログ

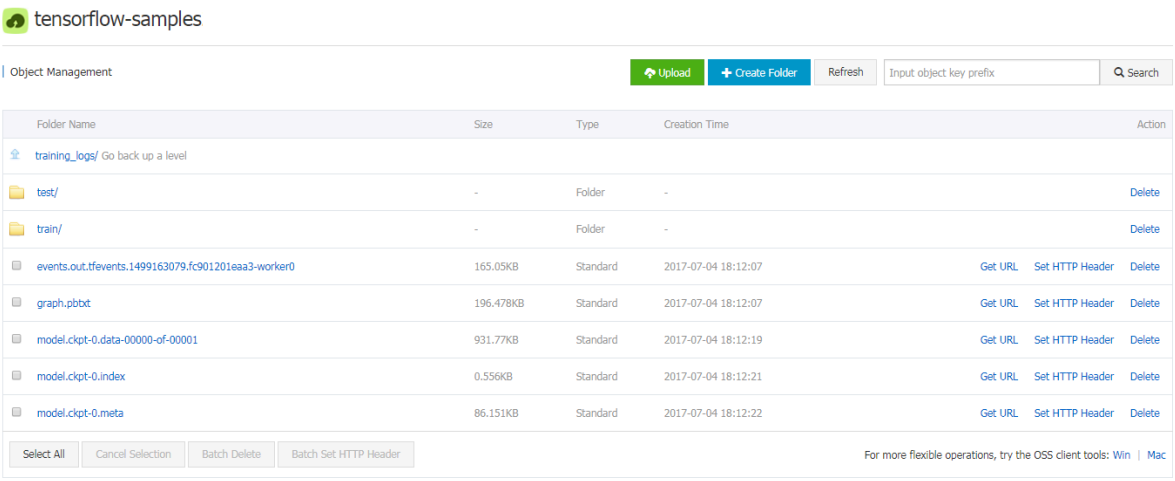


9. [サービス] タブをクリックします。 すべてのワーカーサービスのステータスが [停止] の場合、トレーニングタスクが完了しています。



10.OSS クライアントでは、OSS バケットに自動的に格納されたトレーニング結果を確認できます。

トレーニング完了後、ワーカーコンテナーのローカル / output ディレクトリに保存されたすべてのファイルが、システムにより自動的に指定したお使いのデータボリュームに対応する OSS バケットにコピーされます。 OSS クライアントでは、 / output ディレクトリに書き込まれたイベントファイルおよびチェックポイントファイルが OSS バケットにバックアップされたことを 確認できます。



11.トレーニングタスクを管理します。

トレーニングタスクの停止、再起動または削除するには (たとえば、新しいトレーニングタスクのためにコンピューティングリソースをリリースするため)、[アプリケーション] > [アプリケーションリスト] のページに戻ります。対応するアプリケーションを検索して、右側にある操作を実行します。

6 予測

6.1 TensorFlow Serving の利用

TensorFlow Serving は、Alibaba Cloud エラスティックコンピューティングリソース (ECS (Elastic Compute Service) または EGS)、Server Load Balancer および OSS (Object Storage Service) を使用して TensorFlow モデルに関する予測を行います。

前提条件

モデルトレーニングタスクの実行前に、以下の操作が実行されていることを確認します。

- 一定数のエラスティックコンピューティングリソース (ECS または EGS) を含むコンテナークラスターを作成します。 詳しくは、コンテナークラスターの作成をご参照ください。
- 同一アカウントを使用して、モデルトレーニング用のデータを保存する OSS バケットを作成します。
- 前述のコンテナークラスター用のデータボリュームを作成して、トレーニングタスクを実行するコンテナーにローカルディレクトリとして OSS バケットをマウントします。 詳しくは、データボリュームの作成をご参照ください。
- TensorFlow Serving の基本概念と実行プロセスを理解します。 詳しくは、『*TensorFlow モデルの提供*』をご参照ください。

規則

モデル予測を簡素化するため、以下の規則に注意します。

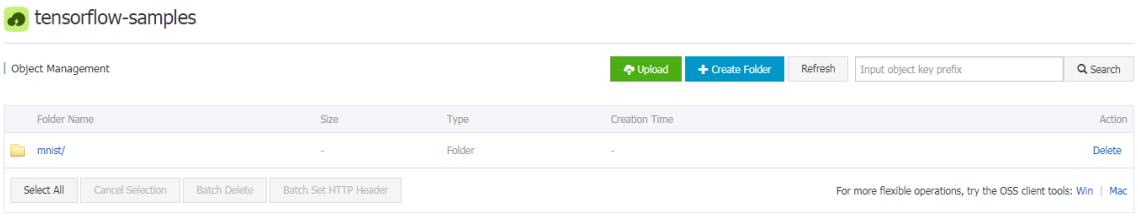
- OSS のルートディレクトリにモデル名を使用したフォルダーを作成すること。
- Server Load Balancer のフロントエンドとバックエンドのポートがモデル予測のアプリケーションポートと一致していることを確認する。

手順

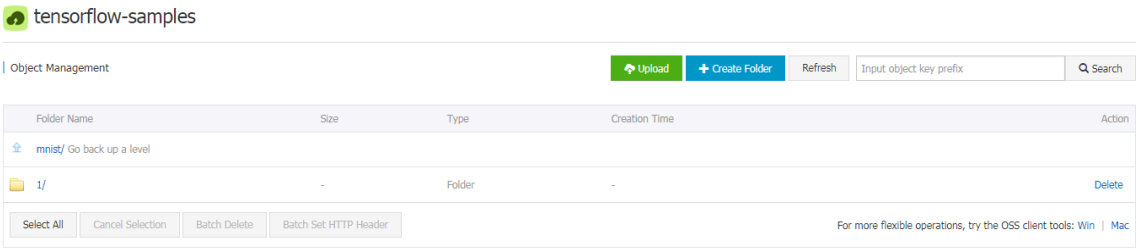
1. 共有ストレージに予測モデルを保存します。

   OSS クライアント上のモデルフォルダーにアップロードすることにより完了させることができます。

   a. OSS バケットのルートディレクトリにモデル名を使用したフォルダーを作成します。 このページの例では、"mnist" を使用します。



   b. TensorFlow Serving モデルフォルダーをバージョン番号を付けて "mnist" にアップロードします。



2. Server Load Balancer のリスニングポートを設定します。

   a. *Server Load Balancer コンソール*で、 右上の [Server Load Balancer の作成] をクリックしてルーティング用の Server Load Balancer インスタンスを作成します。

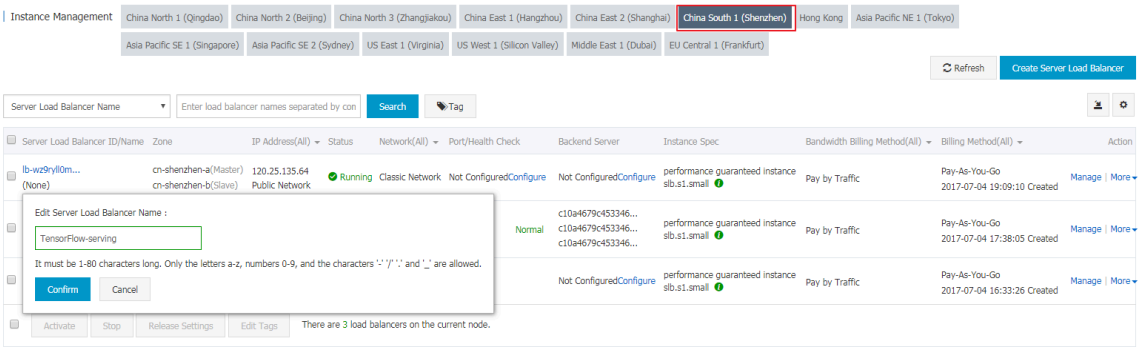
   この例では、インターネット Server Load Balancer インスタンスを作成します。 ニーズに応じて、インターネット Server Load Balancer インスタンスまたはイントラネット Server Load Balancer インスタンスの作成を選択できます。

   > 注 :

使用する Container Service クラスターと同じリージョンを選択します。これは、Server Load Balancer がリージョン間のデプロイをサポートしていないためです。

b. *Server Load Balancer コンソール*に戻り、作成した Server Load Balancer インスタンスの名称を "TensorFlow-serving" と設定します。 Container Service は、この名前を使用して Server Load Balancer インスタンスを参照します。

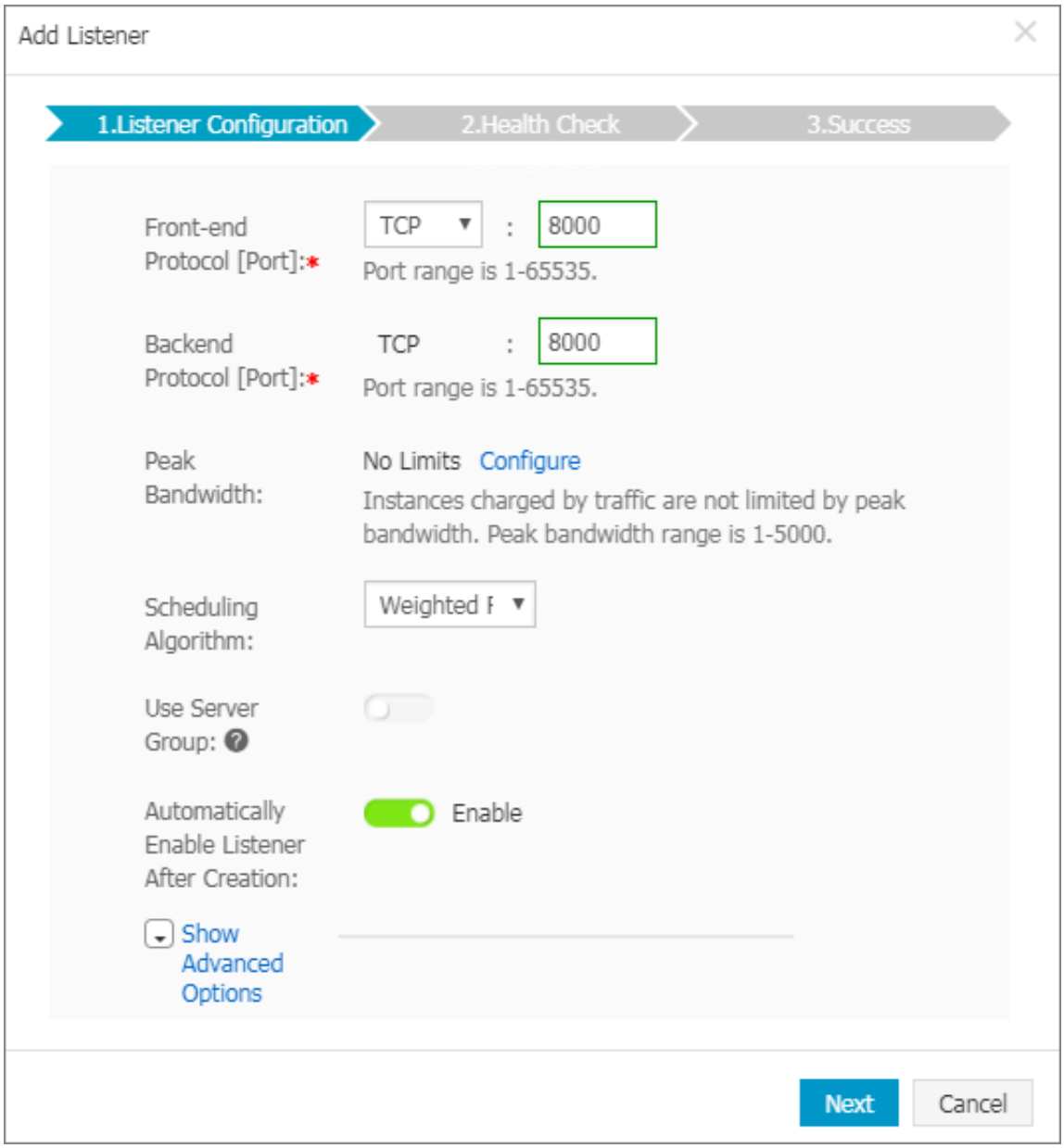
左側のナビゲーションウィンドウから [インスタンス] をクリックします。 Server Load Balancer インスタンスが属するリージョンを選択します。 インスタンス名を編集して [確認] をクリックします。



c. リスニングポートを作成します。

インスタンスの右側にある [管理] をクリックします。 左側のナビゲーションウィンドウから [リスナー] をクリックします。 右上の [リスナーの追加] をクリックして、 設定を行いま

す。 フロントエンドプロトコルとして TCP を選択し、ポートマッピングに "8000:8000" を設定します。

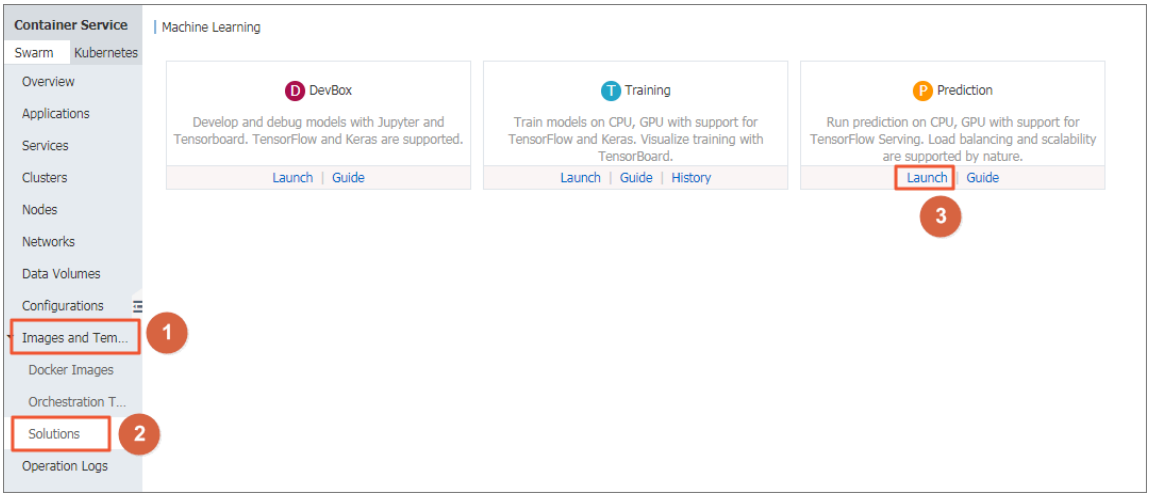


3. モデル予測を開始します。

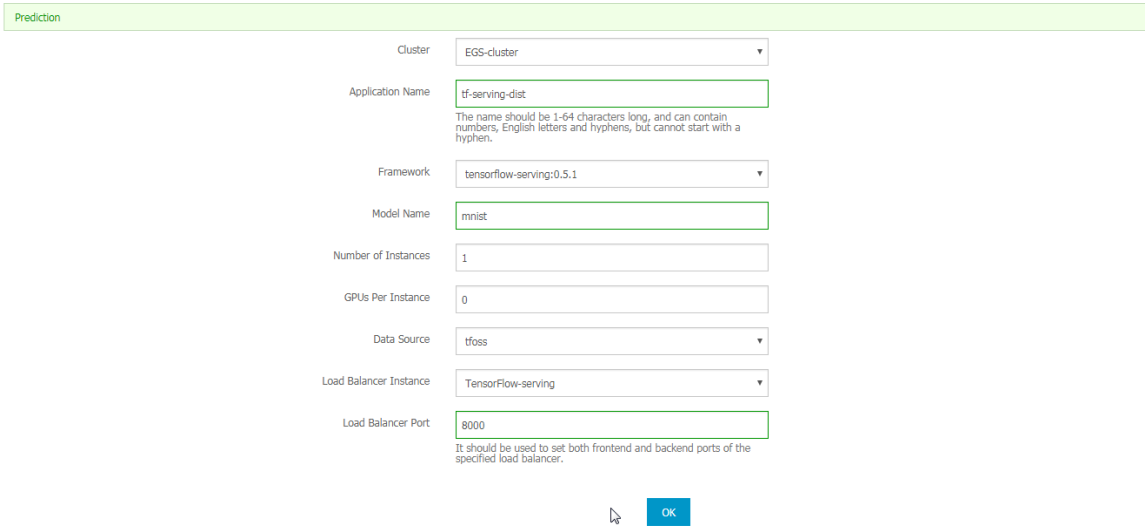
   a. *Container Service コンソール*にログインします。

   b. 左側のナビゲーションウィンドウから [イメージとテンプレート] > > [ソリューション]をクリックします。

   c. [予測] の [起動] をクリックします。



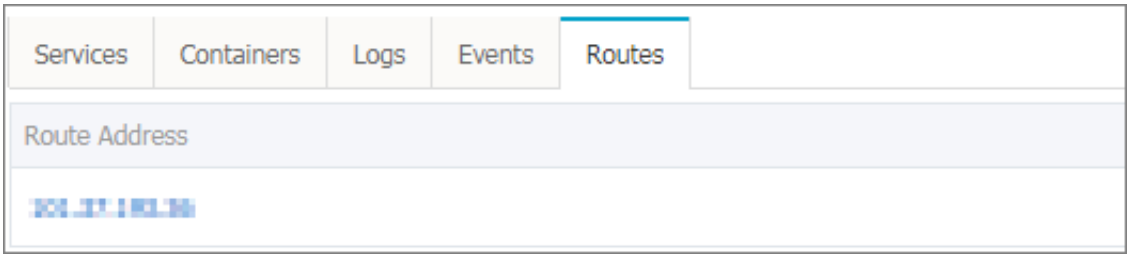
   d. モデル予測タスクに関する基本情報を設定します。



   - クラスター: モデル予測を実行するクラスターを選択します。
   - アプリケーション名: スタンドアロンモデル予測の実行に使用するアプリケーションの名前で、 1 文字以上 64 文字以内で、数字、英字およびハイフン (-) を含みます。
   - フレームワーク: モデルトレーニングに使用するフレームワークを選択します。 TensorFlow Serving およびカスタマイズイメージを含みます。 ここでは、TensorFlow Serving を選択します。

- モデル名: ここでモデル名は手順 1 で作成したモデルフォルダー名と同じものである必要があります。
- インスタンス数: TensorFlow Serving インスタンス数です。 これはクラスターでのノード数を超えることはできません。
- インスタンスごとの GPU: 使用する GPU 数を指定します。 このパラメーターを "0" に設定すると、 GPU の代わりに CPU を使用して予測を行います。
- データソース: 予測モデルを保存するために OSS バケットによりクラスター上に作成されたデータボリュームを 選択します。 データボリュームの作成方法については、データボリュームの作成ご参照ください。
- ロードバランサーインスタンス: 手順 2 で作成した Server Load Balancer インスタンスを選択します。
- ロードバランサーポート: 手順 2 で設定したポートを入力します。

4. 設定完了後、[OK] をクリックします。
5. アプリケーションリストページで、[クラスター] ドロップダウンリストからクラスターを選択して、作成したアプリケーションの名称をクリックします。
6. [ルート] タブをクリックします。 Server Load Balancer が提供する端末アドレスが表示されます。 これにより、gRPC クライアントを使用して "Server Load Balancer アドレス:Server Load Balancer ポート" にアクセスできます。



7 TFRecord および HDFS による TensorFlow トレーニングデータの準備

データの準備と前処理は、ディープラーニングおよびトレーニングプロセスにおいて重要な役割を果たし、モデルトレーニングのスピードと品質に影響します。

TensorFlow は HDFS をサポートし、ビッグデータおよびディープラーニングを統合し、データの準備からモデルトレーニングまでのチェーンを完全なものにします。 Alibaba Cloud Container Service のディープラーニングソリューションは、 TensorFlow をサポートする3 つの分散ストレージバックエンド (OOS (Object Storage Service)、NAS および HDFS) を提供しています。

このドキュメントでは、データの TFRecord 形式への変換方法および、生成された TFRecord ファイルを HDFS に保存する方法を説明します。 この例では、 Alibaba Cloud E-MapReduce (Elastic MapReduce) の HDFS を使っています。

TFRecord を使用する理由

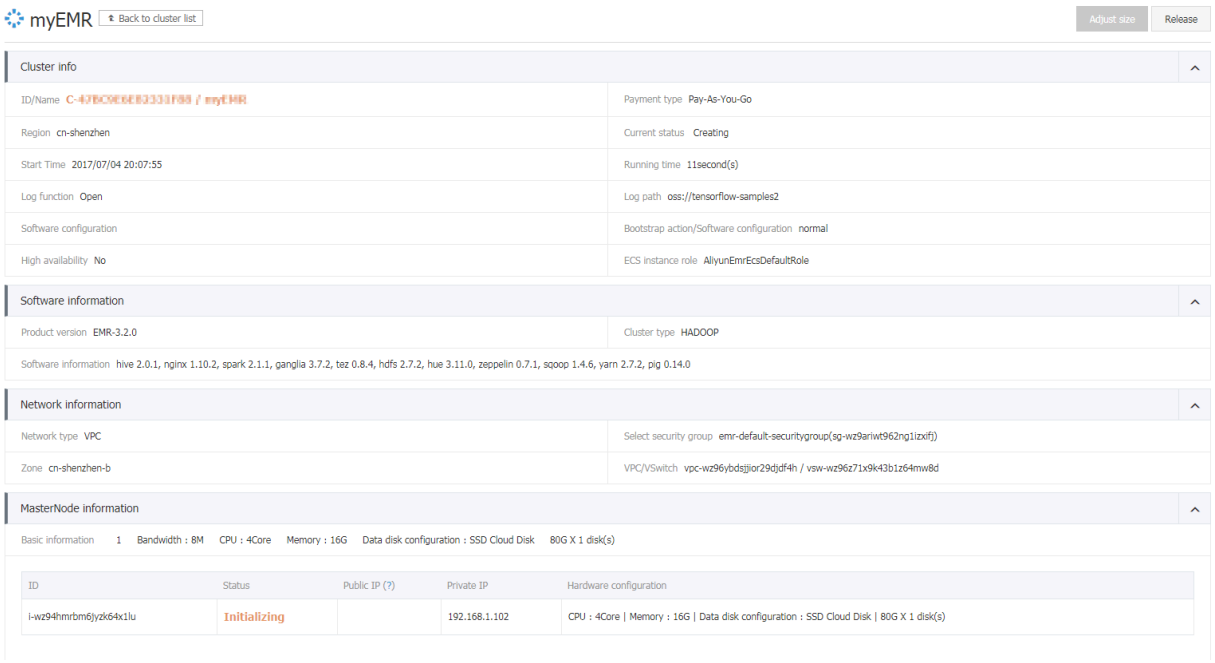
TFRecord は TensorFlow で定義される統一標準データフォーマットです。 マルチスレッドデータの読み込みをサポートし、バッチサイズおよびエポックパラメーターを使用して、 シングルのバッチサイズとトレーニングプロセス中の反復回数を制御します。 TFRecord はメモリーを効率よく使い、データの複製および移動を簡単に行えます。 そのため、TensFlow で大規模なディープラーニングトレーニングを実行する 際の望ましいオプションとなります。

手順 1: E-MapReduce クラスターの作成

E-MapReduce は Alibaba Cloud プラットフォームで動作するビッグデータ処理システムソリューションです。 詳しくは、『*E-MapReduce の概要*』をご参照ください。

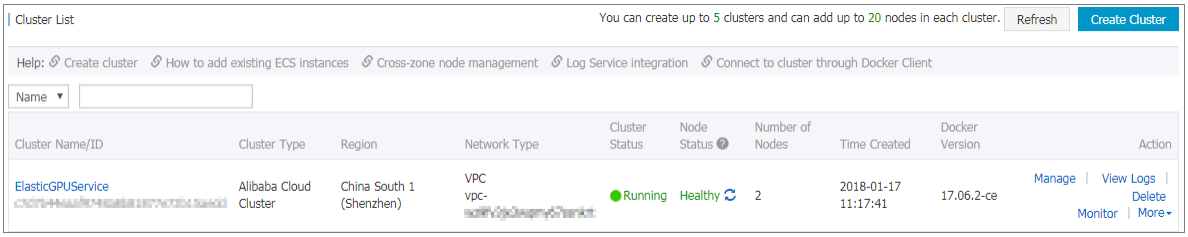
*E-MapReduce コンソール*にログインして E-MapReduce クラスターを 作成します。 E-MapReduce クラスターの作成方法については、『*E-MapReduce クラスターの作成*』をご参照ください。

このページの例では、中国 (深セン) にあるクラスターが作成され、[ネットワークタイプ] が [VPC] に設定されます。



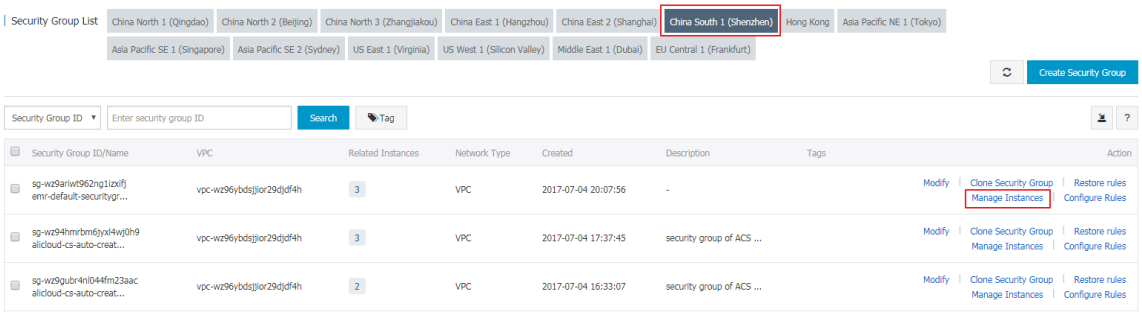
手順 2: コンテナークラスターの作成および 2 つのクラスター間のネットワークの統合

1. *Container Service コンソール*にログインして、同じ VPC (Virtual Private Cloud) 下に GPU コンテナークラスターを作成します。

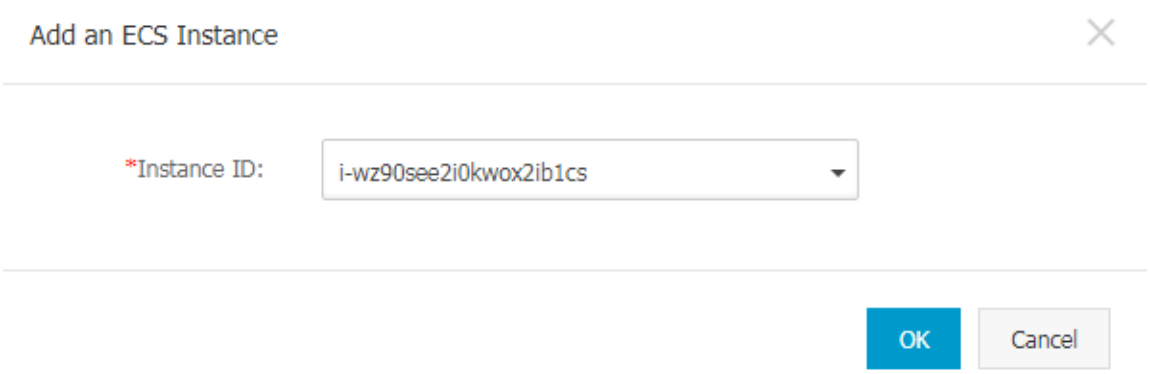


2. *ECS コンソール*にログインして、Container Service クラスターのノードに E-MapReduce クラスターに対応するセキュリティグループを 追加します。

   a. セキュリティグループの属するリージョン (このページの例では、中国 (深セン)) を選択します。 セキュリティグループの右側にある [インスタンスの管理] をクリックします。



   b. 右上の [ECS インスタンスの追加] をクリックします。 コンテナークラスターでノードを選択して、[OK] をクリックします。

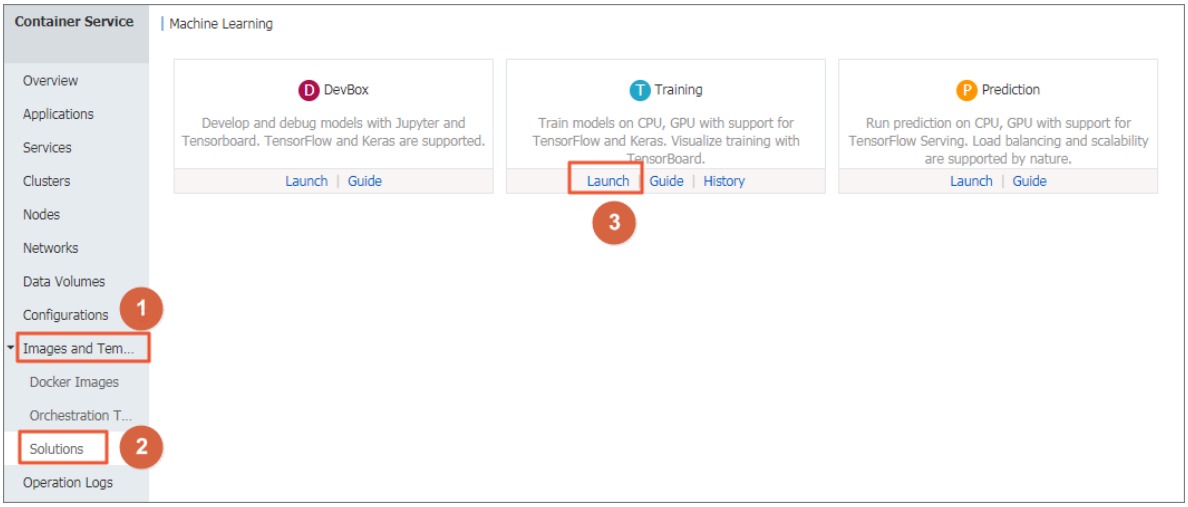


手順 3: TFRecord データの生成

このページの例では、モデルトレーニングサービスにより `convert_to_records.py` を実行する実行環境が提供され、TFRecord データが生成され、HDFS にデータが 保存されます。

1. *Container Service コンソール*にログインします。
2. 左側のナビゲーションウィンドウから [イメージとテンプレート] > [ソリューション] をクリックします。

3. [トレーニング] で [起動] をクリックします。



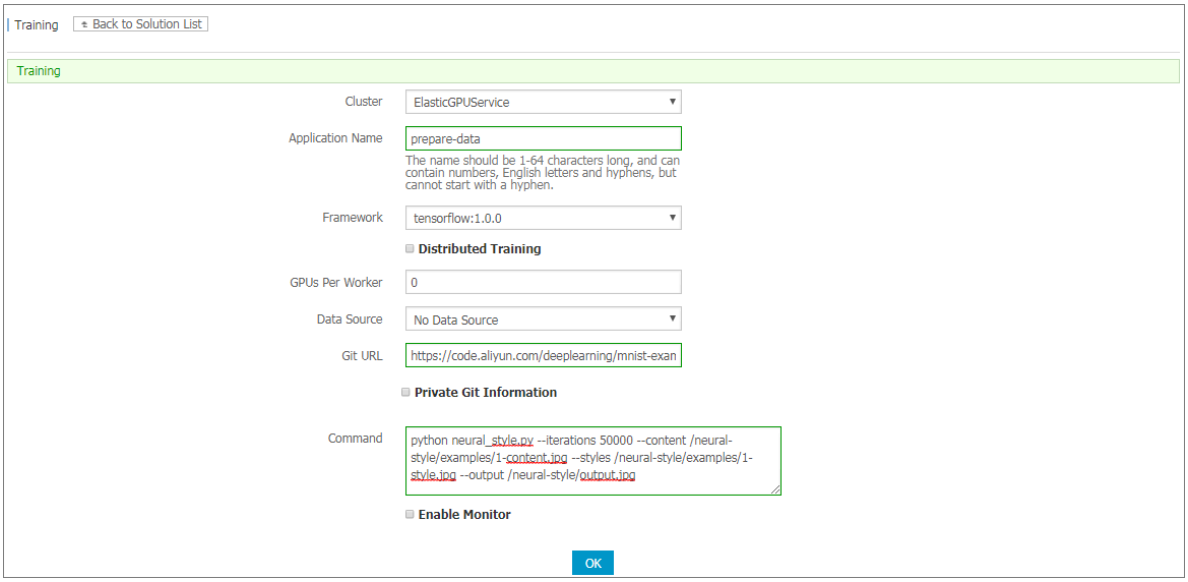
4. モデルトレーニングの設定を行い、[OK] をクリックします。

   このページの例での設定は以下のようになります。

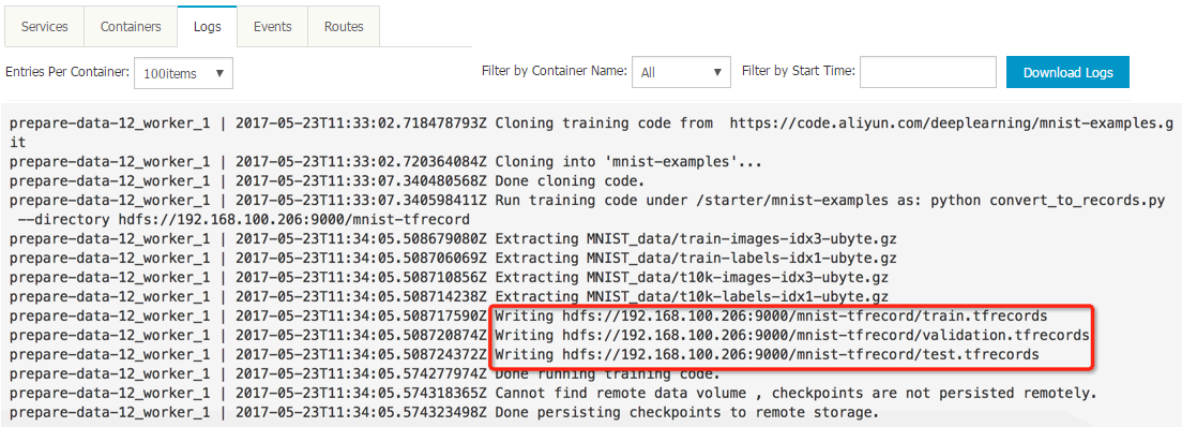
   - フレームワーク: tensorflow:1.0.0 を選択します。
   - ワーカーごとの GPU: "0" を入力します。
   - データソース: "データソースなし" を選択します。
   - Git URL: "https://code.aliyun.com/deeplearning/mnist-examples.git" を入力します。
   - コマンド:

```
python   neural_sty  le . py  -- iterations   50000  -- content
             / neural - style / examples / 1 - content . jpg  --
styles  / neural - style / examples / 1 - style . jpg
```

```
-- output / neural - style / output . jpg
```



これにより、作成したアプリケーションが [アプリケーションリスト] ページに表示されます。アプリケーション名をクリックします。 [ログ] タブをクリックして実行ログを参照します。これは、TFRecord ファイルが HDFS に保存されていることを示しています。



E-MapReduce マシンにログインして、生成された TFRecord ファイルを確認することも できます。

```
# hdfs dfs - ls / mnist - tfrecord
SLF4J : Class path contains multiple SLF4J bindings .
SLF4J : Found binding in [ jar : file :/ opt / apps / hadoop - 2 . 7 . 2 / share / hadoop / common / lib / slf4j - log4j12 - 1 . 7 . 10 . jar ! / org / slf4j / impl / StaticLogg erBinder . class ]
SLF4J : Found binding in [ jar : file :/ opt / apps / tez - 0 . 8 . 4 / lib / slf4j - log4j12 - 1 . 7 . 10 . jar ! / org / slf4j / impl / StaticLogg erBinder . class ]
SLF4J : See http :// www . slf4j . org / codes . html # multiple_b indings for an explanatio n .
SLF4J : Actual binding is of type [ org . slf4j . impl . Log4jLogge rFactory ]
Found 3 items
```

```
- rw - r -- r --  3   root   hadoop   8910000   2017 - 05 - 23   19
 : 34  / mnist - tfrecord / test . tfrecords
- rw - r -- r --  3   root   hadoop   49005000   2017 - 05 - 23   19
 : 33  / mnist - tfrecord / train . tfrecords
- rw - r -- r --  3   root   hadoop   4455000   2017 - 05 - 23   19
 : 33  / mnist - tfrecord / validation . tfrecords
```