aws

开发人员指南

AWS 深度学习容器

AWS 深度学习容器: 开发人员指南

Table of Contents

什么是AWS Deep Learning Containers？

欢迎阅读DeAWS ep Learning Containers 用户指南。

AWSDDeep Learning Containers Leep LDeep Learning Containers Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep Leep TensorFlow TensorFlow PyTorch DDeep Learning Containers Containers Containers 提供具 TensorFlow 有 MXNet、Nvidia CUDA（用于 GPU 实例）和 Intel MKL（用于 CPU 实例）库的优化环境，其在 Amazon Elastic Container Registry (Amazon ECR) gistainer

AWSDeep Learning Containers | Amazon Web Services

关于本指南

本指南可帮助您设置和使用DAWS eep Learning Containers。本指南还介绍如何使用 Amazon EC2、Amazon ECS、Amazon EKS 和设置Deep Learning Containers SageMaker。它介绍了用于培训和推导的深度学习的几种常见使用案例。本指南还为每个框架提供了多个教程。

- 要使用 MXNet、、和 2 在 Amazon EC2 Deep Learning Contain PyTorch er TensorFlow s 上运行训练和 TensorFlow 推理，请参阅Amazon EC2 教程
- 要使用 MXNet 在 Amazon ECS 的Deep Learning Containers 上运行训练和推理 PyTorch TensorFlow、和，请参阅Amazon ECS 教
- 适用于 Amazon EKS 的Deep Learning Containers 提供基于 CPU、GPU 和分布式 GPU 的训练，以及基于 CPU 和 GPU 的推理。要使用 MXNet 在 Amazon EKS 的Deep Learning Containers 上运行训练和推理 PyTorch TensorFlow、和，请参阅Amazon EKS
- 有关基于 Docker 的Deep Learning Containers 镜像、可用镜像列表以及如何使用它们的说明，请参阅Deep Learning Containers 映像
- 有关Deep Learning Containers 安全性的信息，请参阅AWS 深度学习 Containers 中的安全
- 有关最新Deep Learning Containers 发行说明的列表，请参阅深度学习容器的发行说明

Pythththyth

Python 开源社区已于 2020 年 1 月 1 日正式结束对 Python 2 的支持。 TensorFlow 和 PyTorch 社区已经宣布， TensorFlow 2.1 和 PyTorch 1.4 版本将是最后一个支持 Python 2 的版本。支持 Python 2 的Deep Learning Containers 先前版本将继续可用。但是，只有在开源社区发布了 Python 2 Deep

Learning Containers 版本的安全补丁时，我们才会提供这些版本的更新。 TensorFlow 和 PyTorch 框架的下一个版本的Deep Learning Containers 版本将不包括 Python 2 环境。

先决条件

要成功运行Deep Learning Containers，您应该熟悉命令行工具和基本的 Python。有关如何使用每个框架的教程由框架本身提供。但是，本指南向您展示了如何激活每个选项以及如何找到相应的教程以开始使用。

Deep Learning Containers 入门

以下部分介绍了如何使用 Deep Learning Containers 从AWS基础设施。有关将 Deep Learning Containers 与 SageMaker 一起使用的信息，请参阅将您自己的算法或模型与 SageMaker 文档结合使用。

主题

- Amazon EC2 教程
- Amazon ECS 教
- Amazon EKS

Amazon EC2 教程

本部分介绍如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在 EC2 的 Deep Learning Containers 上运行训练和推理。

在开始以下教程之前，请先完成中的步骤Amazon EC2 设置.

目录

- Amazon EC2 设置
- Training
- 推理
- 自定义入口点

Amazon EC2 设置

在本节中，您将了解如何设置AWS使用 Amazon Elastic Compute Cloud 的 Deep Learning Containers。

完成以下步骤以配置您的实例：

- 创建 AWS Identity and Access Management 用户或修改具有下列策略的现有用户。您可以在 IAM 控制台的策略选项卡中按名称搜索策略。
  - AmazonECS_FullAccess 策略
  - AmazonEC2ContainerRegistryFullAccess

有关创建或编辑 IAM 用户的更多信息，请参阅添加和删除 IAM 身份权限在 IAM 用户指南中。

- 启动 Amazon Elastic Compute Cloud 实例（CPU 或 GPU），最好是深度学习基础 AMI. 其他 AMI 工作，但需要相关的 GPU 驱动程序。
- 通过使用 SSH 连接到您的 实例。有关连接的更多信息，请参阅排查实例的连接问题中的Amazon EC2 用户指南。.
- 确保你的AWS CLI使用中的步骤保持最新状态安装最新的AWSCLI 版本.
- 在您的实例中，运行 `aws configure` 并提供已创建用户的凭证。
- 在您的实例中，运行以下命令以登录到托管 Deep Learning Containers 映像的 Amazon ECR 存储库。

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com
```

有关完整列表AWS请参阅 Deep Learning ContainersDeep Learning Containers 映像.

> **Note**
>
> MKL 用户：读取AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议获得最佳训练或推理性能。

后续步骤

要了解有关使用 Deep Learning Containers 在 Amazon EC2 上的训练和推理的信息，请参阅Amazon EC2 教程.

Training

本部分演示如何在上运行训练AWS适用于 Amazon EC2 的 Deep Learning Containers，使用 Apache MXNet、PyTorch、TensorFlow 和 TensorFlow 2 的深度学习容器。

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像.

> **Note**
>
> MKL 用户：读取AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

目录

TensorFlow 训练

在您登录 Amazon EC2 实例后，您可以使用以下命令运行 TensorFlow 和 TensorFlow 2 容器。您必须使用`nvidia-docker`用于 GPU 图像。

- 对于基于 CPU 的培训，请运行以下命令。

```
$ docker run -it <CPU training container>
```

- 对于基于 GPU 的培训，请运行以下命令。

```
$ nvidia-docker run -it <GPU training container>
```

上一命令以交互模式运行容器并在容器内提供一个 shell 提示符。然后，您可以运行以下命令以导入 TensorFlow。

```
$ python
```

```
>> import tensorflow
```

按 Ctrl+D 以返回到 bash 提示符。运行以下命令以开始训练：

```
git clone https://github.com/fchollet/keras.git
```

```
$ cd keras
```

```
$ python examples/mnist_cnn.py
```

后续步骤

要在 Amazon EC2 上使用带有 Deep Learning Containers 的 TensorFlow 了解推理，请参阅TensorFlow 推理.

Apache MXNet（孵化版）训练

要开始利用 Amazon EC2 实例中的 Apache MXNet (孵化) 进行训练，请运行以下命令以运行容器：

- 对于 CPU

```
$ docker run -it <CPU training container>
```

- 对于 GPU

```
$ nvidia-docker run -it <GPU training container>
```

在容器的终端中，运行以下命令以开始训练。

- 对于 CPU

```
$ git clone -b v1.4.x https://github.com/apache/incubator-mxnet.git
python incubator-mxnet/example/image-classification/train_mnist.py
```

- 对于 GPU

```
$ git clone -b v1.4.x https://github.com/apache/incubator-mxnet.git
python incubator-mxnet/example/image-classification/train_mnist.py --gpus 0
```

使用 gluonCV 进行 MxNet 培训

在容器的终端中，运行以下命令以开始使用 GluonCV 进行训练。Deep Learning Containers 中包含 GluonCV v0.6.0。

- 对于 CPU

```
$ git clone -b v0.6.0 https://github.com/dmlc/gluon-cv.git
python gluon-cv/scripts/classification/cifar/train_cifar10.py --model resnet18_v1b
```

- 对于 GPU

```
$ git clone -b v0.6.0 https://github.com/dmlc/gluon-cv.git
python gluon-cv/scripts/classification/cifar/train_cifar10.py --num-gpus 1 --model
 resnet18_v1b
```

后续步骤

要在 Amazon EC2 上使用 MxNet 与 Deep Learning Containers 一起学习推理，请参阅推理 Apache MXNet（孵化）.

PyTorch 训练

要开始利用 Amazon EC2 实例进行 PyTorch 训练，请使用以下命令来运行容器。您必须使用**nvidia-docker**用于 GPU 图像。

- 对于 CPU

```
$ docker run -it <CPU training container>
```

- 对于 GPU

```
$ nvidia-docker run -it <GPU training container>
```

- 如果您的 docker-ce 版本 19.03 或更高版本，则可以在 docker 中使用 —gpus 标志：

```
$ docker run -it --gpus <GPU training container>
```

运行以下命令以开始训练。

- 对于 CPU

```
$ git clone https://github.com/pytorch/examples.git
$ python examples/mnist/main.py --no-cuda
```

- 对于 GPU

```
$ git clone https://github.com/pytorch/examples.git
$ python examples/mnist/main.py
```

PyTorch 利用 NVIDIA Apex 分发了 GPU 培训

NVIDIA Apex 是 PyTorch 扩展程序，其实用程序可用于混合精度和分布式培训。有关 Apex 提供的实用程序的更多信息，请参阅NVIDIA 顶级网站. 以下系列中的 Amazon EC2 实例支持 Apex：

- Amazon EC2 P3 实例
- Amazon EC2 P2 实例
- Amazon EC2 G4 实例
- Amazon EC2 G3 实例

要使用 NVIDIA Apex 开始分布式训练，请在 GPU 训练容器的终端中运行以下命令。此示例需要在 Amazon EC2 实例上至少有两个 GPU 才能运行并行分布式培训。

```
$ git clone https://github.com/NVIDIA/apex.git && cd apex
$ python -m torch.distributed.launch --nproc_per_node=2 examples/simple/distributed/
distributed_data_parallel.py
```

PyTorch 的 Amazon S3 插件

Deep Learning Containers 包括一个插件，使您能够将 Amazon S3 存储桶中的数据用于 PyTorch 培训。

1. 要开始在 Deep Learning Containers 中使用 Amazon S3 插件，请检查以确保您的 Amazon EC2 实例具有对 Amazon S3 的完全访问权限。创建 IAM 角色授予 Amazon S3 访问 Amazon EC2 实例的权限并将该角色附加到您的实例。您可以使用AmazonS3FullAccess要么AmazonS3ReadOnlyAccess政策。
2. 设置您的`AWS_REGION`环境变量与您选择的区域。

   ```
   export AWS_REGION=us-east-1
   ```

3. 使用以下命令运行与 Amazon S3 插件兼容的容器。您必须使用**`nvidia-docker`**用于 GPU 图像。

   - 对于 CPU

```
docker run -it 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.8.1-cpu-py36-ubuntu18.04-v1.6
```

- 对于 GPU

```
nvidia-docker run -it 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.8.1-gpu-py36-cu111-ubuntu18.04-v1.7
```

4. 运行以下命令来测试示例。

```
git clone https://github.com/aws/amazon-s3-plugin-for-pytorch.git
cd amazon-s3-plugin-for-pytorch/examples
python s3_cv_iterable_shuffle_example.py
```

有关更多信息和其他示例，请参阅PyTorch 的 Amazon S3 插件存储库。

后续步骤

要在 Amazon EC2 上使用 PyTorch 与 Deep Learning Containers 进行推理，请参阅PyTorch 推理 .

推理

此节演示如何在上运行推理AWS亚马逊弹性计算云的 Deep Learning Containers 使用 Apache MxNet（孵育）、PyTorch、TensorFlow 和 TensorFlow 2. 你也可以使用 Elastic Inference 来运行推理 AWSDeep Learning Containers。有关 Elastic Inference 的教程和更多信息，请参阅使用AWSAmazon EC2 上的 EElastic Inference Deep Learning Containers.

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像.

> **Note**
>
> MKL 用户：读取 AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

目录

- 推理 Apache MXNet（孵化）
- PyTorch 推理

TensorFlow 推理

为了演示如何使用 Deep Learning Containers 进行推理，本示例使用了一个简单的一半加两模型 TensorFlow 服务。我们建议使用深度学习基础 AMI对于 TensorFlow。登录实例后，运行以下命令：

```
$ git clone -b r1.15 https://github.com/tensorflow/serving.git
$ cd serving
$ git checkout r1.15
```

使用这里的命令开始 TensorFlow 与此模型的 Deep Learning Containers 一起服务。请注意，与用于训练的不同，模型处理将在运行容器后立即开始并且作为后台进程运行。

- 对于 CPU 实例：

```
$ docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --mount
 type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_cpu,target=/models/saved_model_half_plus_two -e
 MODEL_NAME=saved_model_half_plus_two -d <cpu inference container>
```

  例如：

```
$ docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --mount
 type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_cpu,target=/models/saved_model_half_plus_two
 -e MODEL_NAME=saved_model_half_plus_two -d 763104351884.dkr.ecr.us-
east-1.amazonaws.com/tensorflow-inference:1.15.0-cpu-py36-ubuntu18.04
```

- 对于 GPU 实例：

```
$ nvidia-docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --
mount type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_gpu,target=/models/saved_model_half_plus_two -e
 MODEL_NAME=saved_model_half_plus_two -d <gpu inference container>
```

  例如：

```
$ nvidia-docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference
 --mount type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/
testdata/saved_model_half_plus_two_gpu,target=/models/saved_model_half_plus_two
 -e MODEL_NAME=saved_model_half_plus_two -d 763104351884.dkr.ecr.us-
east-1.amazonaws.com/tensorflow-inference:1.15.0-gpu-py36-cu100-ubuntu18.04
```

- 对于 Inf1 实例：

```
$ docker run -id --name tensorflow-inference -p 8500:8500 --device=/dev/neuron0  --
cap-add IPC_LOCK  --mount type=bind,source={model_path},target=/models/{model_name}
 -e MODEL_NAME={model_name} <neuron inference container>
```

例如：

```
$ docker run -id --name tensorflow-inference -p 8500:8500 --device=/dev/neuron0  --
cap-add IPC_LOCK  --mount type=bind,source={model_path},target=/models/{model_name}
 -e MODEL_NAME={model_name} 763104351884.dkr.ecr.us-west-2.amazonaws.com/tensorflow-
inference-neuron:1.15.4-neuron-py37-ubuntu18.04-v1.1
```

接下来，使用 Deep Learning Containers 运行推理。

```
$ curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST http://127.0.0.1:8501/v1/models/
saved_model_half_plus_two:predict
```

输出类似于以下内容：

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

> **Note**
>
> 如果您想要调试容器的输出，可以使用容器名称来附加输出，可以使用容器名称来附加输出：
>
> ```
> $ docker attach <your docker container name>
> ```
>
> 在这个例子中你使用tensorflow-inference.

TensorFlow 2 推理

为了演示如何使用 Deep Learning Containers 进行推理，本示例使用了一个简单的一半加两模型 TensorFlow 2 个服务。我们建议使用深度学习基础 AMI为了 TensorFlow 2. 登录实例后，运行以下命令。

```
$ git clone -b r2.0 https://github.com/tensorflow/serving.git
$ cd serving
```

使用这里的命令开始 TensorFlow 与此模型的 Deep Learning Containers 一起服务。请注意，与用于训练的不同，模型处理将在运行容器后立即开始并且作为后台进程运行。

- 对于 CPU 实例：

```
$ docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --mount
 type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_cpu,target=/models/saved_model_half_plus_two -e
 MODEL_NAME=saved_model_half_plus_two -d <cpu inference container>
```

  例如：

```
$ docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --mount
 type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_cpu,target=/models/saved_model_half_plus_two
 -e MODEL_NAME=saved_model_half_plus_two -d 763104351884.dkr.ecr.us-
east-1.amazonaws.com/tensorflow-inference:2.0.0-cpu-py36-ubuntu18.04
```

- 对于 GPU 实例：

```
$ nvidia-docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference --
mount type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/testdata/
saved_model_half_plus_two_gpu,target=/models/saved_model_half_plus_two -e
 MODEL_NAME=saved_model_half_plus_two -d <gpu inference container>
```

  例如：

```
$ nvidia-docker run -p 8500:8500 -p 8501:8501 --name tensorflow-inference
 --mount type=bind,source=$(pwd)/tensorflow_serving/servables/tensorflow/
testdata/saved_model_half_plus_two_gpu,target=/models/saved_model_half_plus_two
```

```
 -e MODEL_NAME=saved_model_half_plus_two -d 763104351884.dkr.ecr.us-
east-1.amazonaws.com/tensorflow-inference:2.0.0-gpu-py36-cu100-ubuntu18.04
```

> **Note**
>
> 加载 GPU 模型服务器可能需要一段时间。

接下来，使用 Deep Learning Containers 运行推理。

```
$ curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST http://127.0.0.1:8501/v1/models/
saved_model_half_plus_two:predict
```

输出类似于以下内容。

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

> **Note**
>
> 要调试容器的输出，可以使用名称来附加容器的输出，如以下命令所示：
>
> ```
> $ docker attach <your docker container name>
> ```
>
> 使用此示例tensorflow-inference.

推理 Apache MXNet（孵化）

要开始使用 Apache MXNet 存储桶进行推理，此示例使用来自公有 S3 存储桶的一个预先训练好的模型。

对于 CPU 实例，运行以下命令。

```
$ docker run -it --name mms -p 80:8080  -p 8081:8081 <your container image id> \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
```

```
--models squeezenet=https://s3.amazonaws.com/model-server/models/squeezenet_v1.1/
squeezenet_v1.1.model
```

对于 GPU 实例，运行以下命令：

```
$ nvidia-docker run -it --name mms -p 80:8080  -p 8081:8081 <your container image id> \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
--models squeezenet=https://s3.amazonaws.com/model-server/models/squeezenet_v1.1/
squeezenet_v1.1.model
```

该配置文件包含在容器中。

在启动您的服务器后，现在您可以使用以下命令从不同的窗口来运行推理。

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl -X POST http://127.0.0.1/predictions/squeezenet -T kitten.jpg
```

在您使用完容器后，可以使用以下命令将其删除：

```
$ docker rm -f mms
```

GluonCV 的 MXNet 推理

要开始使用 GluonCV 进行推理，此示例使用来自公有 S3 存储桶的一个预先训练好的模型。

对于 CPU 实例，运行以下命令。

```
$ docker run -it --name mms -p 80:8080  -p 8081:8081 <your container image id> \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
--models gluoncv_yolo3=https://dlc-samples.s3.amazonaws.com/mxnet/gluon/
gluoncv_yolo3.mar
```

对于 GPU 实例，运行以下命令。

```
$ nvidia-docker run -it --name mms -p 80:8080  -p 8081:8081 <your container image id> \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
--models gluoncv_yolo3=https://dlc-samples.s3.amazonaws.com/mxnet/gluon/
gluoncv_yolo3.mar
```

该配置文件包含在容器中。

在启动您的服务器后，现在您可以使用以下命令从不同的窗口来运行推理。

```
$ curl -O https://dlc-samples.s3.amazonaws.com/mxnet/gluon/dog.jpg
curl -X POST http://127.0.0.1/predictions/gluoncv_yolo3/predict -T dog.jpg
```

您的输出应与以下内容类似：

```
{
  "bicycle": [
    "[ 79.674225  87.403786 409.43515  323.12167 ]",
    "[ 98.69891  107.480446 200.0086   155.13412 ]"
  ],
  "car": [
    "[336.61322   56.533463 499.30566  125.0233  ]"
  ],
  "dog": [
    "[100.50538 156.50375 223.014   384.60873]"
  ]
}
```

在您使用完容器后，可以使用此命令将其删除。

```
$ docker rm -f mms
```

PyTorch 推理

具有 Deep Learning Containers PyTorch 版本 1.6 及更高版本 TorchServe 用于推理调用。具有 Deep Learning Containers PyTorch 版本 1.5 及更低版本使用`mxnet-model-server`用于推理调用。

PyTorch 1.6 及更高版本

要使用 PyTorch 运行推理，此示例使用来自公有 S3 存储桶的在 Imageet 上预先训练的模型。推理是使用 TorchServe 提供的。有关更多信息，请参阅上的此博客部署 PyTorch 使用 TorchServe 进行推理.

对于 CPU 实例：

```
$ docker run -itd --name torchserve -p 80:8080  -p 8081:8081 <your container image id>
 \
torchserve --start --ts-config /home/model-server/config.properties \
--models pytorch-densenet=https://torchserve.s3.amazonaws.com/mar_files/densenet161.mar
```

对于 GPU 实例

```
$ nvidia-docker run -itd --name torchserve -p 80:8080  -p 8081:8081 <your container
 image id> \
torchserve --start --ts-config /home/model-server/config.properties \
--models pytorch-densenet=https://torchserve.s3.amazonaws.com/mar_files/densenet161.mar
```

如果您的 docker-ce 版本 19.03 或更高版本，则可以使用--gpus启动 Docker 时标记。

该配置文件包含在容器中。

在启动您的服务器后，现在您可以使用以下命令从不同的窗口来运行推理。

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://127.0.0.1:80/predictions/pytorch-densenet -T flower.jpg
```

在您使用完容器后，可以使用以下命令将其删除。

```
$ docker rm -f torchserve
```

PyTorch 1.5 及更低版本

要使用 PyTorch 运行推理，此示例使用来自公有 S3 存储桶的在 Imageet 上预先训练的模型。与 MXNet 容器类似，使用 mxnet-model-server 提供推理，该服务器可以支持任何框架作为后端。有关更多信息，请参阅 。适用于 Apache MXNet 的模型服务器还有这个博客部署 PyTorch 推理 MXNet 模型服务器.

对于 CPU 实例：

```
$ docker run -itd --name mms -p 80:8080  -p 8081:8081 <your container image id> \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
--models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/multi-model-server/
densenet/densenet.mar
```

对于 GPU 实例

```
$ nvidia-docker run -itd --name mms -p 80:8080  -p 8081:8081 <your container image id>
 \
mxnet-model-server --start --mms-config /home/model-server/config.properties \
--models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/multi-model-server/
densenet/densenet.mar
```

如果您的 docker-ce 版本 19.03 或更高版本，则可以使用--gpus启动 Docker 时标记。

该配置文件包含在容器中。

在启动您的服务器后，现在您可以使用以下命令从不同的窗口来运行推理。

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://127.0.0.1/predictions/densenet -T flower.jpg
```

在您使用完容器后，可以使用以下命令将其删除。

```
$ docker rm -f mms
```

后续步骤

要了解如何在 Amazon ECS 上将自定义入口点与 Deep Learning Containers 结合使用，请参阅自定义入口点.

自定义入口点

对于某些映像，Deep Learning Containers 使用自定义入口点脚本。如果您要使用自己的入口点，可以按如下方式覆盖入口点。

- 要指定要运行的自定义入口点脚本，请使用此命令。

  ```
  docker run --entrypoint=/path/to/custom_entrypoint_script -it <image> /bin/bash
  ```

- 要将入口点设置为空，请使用此命令。

  ```
  docker run --entrypoint="" <image> /bin/bash
  ```

Amazon ECS 教

本部分介绍如何使用 MXNet、和，在适用于 Amazon ECS 的DeeAWS p Learning Containers 上运行训练和推理 TensorFlow。 PyTorch

在开始以下教程之前，请先完成中的步骤Amazon ECS 设置。

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像。

> **Note**
>
> MKL 用户：阅读AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

目录

Amazon ECS 设置

本主题介绍如何使用 Amazon 弹性容器服务设置AWS Deep Learning Containers。

目录

先决条件

本安装指南假设您已完成以下先决条件：

- 安装和配置最新版本的AWS CLI。有关安装或升级 AWS CLI 的更多信息，请参阅安装 AWS Command Line Interface。
- 完成使用 Amazon ECS 进行设置中的步骤。
- 确认您拥有 Amazon ECS 容器实例角色。有关更多信息，请参阅 A mazon ECS 容器服务开发人员指南中的 Amazon ECS 容器实例 IAM 角色。
- Amazon CloudWatch Logs IAM 策略已添加到亚马逊 ECS 容器实例角色中，该角色允许 Amazon ECS 向亚马逊发送日志 CloudWatch。有关更多信息，请参阅 A mazon Elastic Container Service 开发人员指南中的CloudWatch L ogs IAM
- 创建新的安全组或更新现有的安全组，以便为所需的推理服务器打开端口。
  - 对于 MXNet 推断，端口 80 和 8081 向 TCP 流量开放。
  - 为了进行 TensorFlow 推断，端口 8501 和 8500 向 TCP 流量开放。

有关更多信息，请参阅 Amazon EC2 安全组。

为Deep Learning Containers 设置 Amazon ECS

本部分介绍如何设置 Amazon ECS 以使用Deep Learning Containers。

> **Important**
>
> 如果您的账户已创建 Amazon ECS 服务相关角色，则默认情况下会为您的服务使用该角色，除非您在此处指定一个角色。如果任务定义使用 awsvpc 网络模式，或者将服务配置为使用以下任一内容：服务发现、外部部署控制器、多个目标组或 Elastic Inference 加速器（Service 加速器），则需要使用服务相关角色。如果是这种情况，则不应在此处指定角色。有关更多信息，请参阅 Amazon ECS 开发人员指南中的 Amazon ECS 使用服务关联角色。

从您的主机运行以下操作。

1. 在包含您之前创建的key pair 和安全组的区域中创建一个 Amazon ECS 集群。

```
aws ecs create-cluster --cluster-name ecs-ec2-training-inference --region us-east-1
```

2. 将一个或多个 Amazon EC2 实例启动到您的集群中。有关基于 GPU 的工作，请参阅《亚马逊 ECS 开发者指南》中的“在 Amazon ECS 上使用 GPU”，以告知您的实例类型选择。如果选择 GPU 实例类型，请确保随后选择 Amazon ECS 经 GPU 优化的 AMI。对于基于 CPU 的工作，可以使用 Amazon Linux 或 Amazon Linux 2 ECS 优化的 AMI。有关兼容实例类型和 Amazon ECS ECS 优化的 AMI ID ID 的更多信息，请参阅经 Amazon ECS 优化的 AMI。在本示例中，您在 us-east-1 中使用基于 GPU 的 AMI 启动一个实例，磁盘大小为 100 GB。

   a. 使用以下内容创建名为 `my_script.txt` 的文件。引用您在上一步中创建的同一集群名称。

```
#!/bin/bash
echo ECS_CLUSTER=ecs-ec2-training-inference >> /etc/ecs/ecs.config
```

   b. （可选）使用以下内容创建名为 `my_mapping.txt` 的文件，这将在创建实例后更改根卷的大小。

```
[
    {
```

```
        "DeviceName": "/dev/xvda",
        "Ebs": {
            "VolumeSize": 100
        }
    }
]
```

c. 启动一个带有 Amazon ECS ECS 优化的 AMI 实例并将其连接到集群。使用您创建的安全组 ID 和key pair 名称，并在以下命令中替换它们。要获取最新的经 Amazon ECS 优化的 AMI ID ID，请参阅 Amazon Elastic ContainECS 开发人员指南中的 Amazon EC S 优化的 AMI ID ID。

```
aws ec2 run-instances --image-id ami-0dfdeb4b6d47a87a2 \
                      --count 1 \
                      --instance-type p2.8xlarge \
                      --key-name key-pair-1234 \
                      --security-group-ids sg-abcd1234 \
                      --iam-instance-profile Name="ecsInstanceRole" \
                      --user-data file://my_script.txt \
                      --block-device-mapping file://my_mapping.txt \
                      --region us-east-1
```

在 Amazon EC2 控制台中，您可以通过响应instance-id中的，验证本步骤是否成功。

您现在有一个正在运行容器实例的 Amazon ECS 集群。通过以下步骤验证 Amazon EC2 实例是否已在集群中注册。

验证 Amazon EC2 实例已在集群中注册

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
2. 选择包含您已注册 Amazon EC2 实例的集群。
3. 在集群页面上，选择基础设施。
4. 在容器实例下，验证是否显示instance-id了在上一步中创建的。另外，请注意可用 CPU 和可用内存的值，因为这些值在以下教程中可能很有用。上述值可能需要几分钟才能显示在控制台中。

后续步骤

要了解有关在 Amazon ECS 上使用Deep Learning Containers 进行训练和推理的信息，请参阅Amazon ECS 教。

培训

本节介绍如何使用 Apache MXNet（孵化）、、和 TensorFlow 2 在适用于 Amazon Elastic 容器服务的DeeAWS p Learning Conta PyTorch in TensorFlow ers 上运行训练。

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像。

> **Note**
>
> MKL 用户：阅读AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

> **Important**
>
> 如果您的账户已创建 Amazon ECS 服务相关角色，则默认情况下会为您的服务使用该角色，除非您在此处指定一个角色。如果任务定义使用 awsvpc 网络模式，或者将服务配置为使用服务发现，则需要使用服务相关角色。如果服务使用外部部署控制器、多个目标组或 Elastic Inference 加速器（在这种情况下，不应在此处指定角色），则还需要使用该角色。有关更多信息，请参阅 Amazon ECS 开发人员指南中的 Amazon ECS 使用服务关联角色。

目录

TensorFlow训练

您必须先注册任务定义才能在 ECS 集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用了向Deep Learning Containers 添加训练脚本的 Docker 示例。您可以将此脚本与任一 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 映像。

1. 使用以下内容创建名为 ecs-deep-learning-container-training-taskdef.json 的文件。

   - 适用于 CPU

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mkdir -p /test && cd /test && git clone https://github.com/fchollet/keras.git
 && chmod +x -R /test/ && python keras/examples/mnist_cnn.py"
  ],
  "entryPoint": [
   "sh",
   "-c"
  ],
  "name": "tensorflow-training-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.2-cpu-py36-ubuntu18.04",
  "memory": 4000,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "protocol": "tcp"
  }],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "awslogs-tf-ecs",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "tf",
    "awslogs-create-group": "true"
   }
  }
 }],
 "volumes": [],
 "networkMode": "bridge",
 "placementConstraints": [],
 "family": "TensorFlow"
}
```

- 对于 GPU

```
{
      "requiresCompatibilities": [
        "EC2"
      ],
      "containerDefinitions": [
        {
    "command": [
                "mkdir -p /test && cd /test && git clone https://github.com/
fchollet/keras.git && chmod +x -R /test/ && python keras/examples/mnist_cnn.py"
             ],
             "entryPoint": [
                "sh",
                "-c"
             ],
          "name": "tensorflow-training-container",
          "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
training:1.15.2-gpu-py37-cu100-ubuntu18.04",
          "memory": 6111,
          "cpu": 256,
          "resourceRequirements" : [{
            "type" : "GPU",
            "value" : "1"
          }],
          "essential": true,
          "portMappings": [
            {
              "containerPort": 80,
              "protocol": "tcp"
            }
          ],
          "logConfiguration": {
              "logDriver": "awslogs",
              "options": {
                  "awslogs-group": "awslogs-tf-ecs",
                  "awslogs-region": "us-east-1",
                  "awslogs-stream-prefix": "tf",
          "awslogs-create-group": "true"
              }
          }
        }
      ],
```

```
      "volumes": [],
      "networkMode": "bridge",
      "placementConstraints": [],
      "family": "tensorflow-training"
    }
```

2. 注册任务定义。请记下输中的修订号，并在下这些中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-deep-learning-container-training-taskdef.json
```

3. 使用任务定义创建任务。您需要上一步的修订号和在安装期间创建的集群的名称

```
aws ecs run-task --cluster ecs-ec2-training-inference --task-definition tf:1
```

4. 在 https://console.aws.amazon.com/ecs/ 上打开 AmazECS 经典控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 任务处于某种RUNNING状态后，选择任务标识符。
8. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

后续步骤

要使用D TensorFlow eep Learning Containers 在 Amazon ECS 上学习推理，请参阅TensorFlow理推理推理。

Apache MXNet（孵化）培训

您必须先注册任务定义才能在 AmazECECECON 上运行任务。任务定义是分组在一起的一系列容器。以下示例使用了向Deep Learning Containers 添加训练脚本的 Docker 示例。

1. 使用以下内容创建名为 ecs-deep-learning-container-training-taskdef.json 的文件。

   - 适用于 CPU

```
{
   "requiresCompatibilities":[
      "EC2"
```

```
   ],
   "containerDefinitions":[
      {
         "command":[
            "git clone -b 1.4 https://github.com/apache/incubator-mxnet.git &&
 python /incubator-mxnet/example/image-classification/train_mnist.py"
         ],
         "entryPoint":[
            "sh",
            "-c"
         ],
         "name":"mxnet-training",
         "image":"763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-
training:1.6.0-cpu-py36-ubuntu16.04",
         "memory":4000,
         "cpu":256,
         "essential":true,
         "portMappings":[
            {
               "containerPort":80,
               "protocol":"tcp"
            }
         ],
         "logConfiguration":{
            "logDriver":"awslogs",
            "options":{
               "awslogs-group":"/ecs/mxnet-training-cpu",
               "awslogs-region":"us-east-1",
               "awslogs-stream-prefix":"mnist",
               "awslogs-create-group":"true"
            }
         }
      }
   ],
   "volumes":[

   ],
   "networkMode":"bridge",
   "placementConstraints":[

   ],
   "family":"mxnet"
}
```

- 对于 GPU

```
{
   "requiresCompatibilities":[
      "EC2"
   ],
   "containerDefinitions":[
      {
         "command":[
            "git clone -b 1.4 https://github.com/apache/incubator-mxnet.git &&
 python /incubator-mxnet/example/image-classification/train_mnist.py --gpus 0"
         ],
         "entryPoint":[
            "sh",
            "-c"
         ],
         "name":"mxnet-training",
         "image":"763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-
training:1.6.0-gpu-py36-cu101-ubuntu16.04",
         "memory":4000,
         "cpu":256,
         "resourceRequirements":[
            {
               "type":"GPU",
               "value":"1"
            }
         ],
         "essential":true,
         "portMappings":[
            {
               "containerPort":80,
               "protocol":"tcp"
            }
         ],
         "logConfiguration":{
            "logDriver":"awslogs",
            "options":{
               "awslogs-group":"/ecs/mxnet-training-gpu",
               "awslogs-region":"us-east-1",
               "awslogs-stream-prefix":"mnist",
               "awslogs-create-group":"true"
            }
         }
```

```
        }
    ],
    "volumes":[

    ],
    "networkMode":"bridge",
    "placementConstraints":[

    ],
    "family":"mxnet-training"
}
```

2. 注册任务定义。请记下输中的修订号，并在下这些中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-deep-learning-container-training-taskdef.json
```

3. 使用任务定义创建任务。您需要上中的修订号。

```
aws ecs run-task --cluster ecs-ec2-training-inference --task-definition mx:1
```

4. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 任务处于某种RUNNING状态后，选择任务标识符。
8. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

后续步骤

要使用带有Deep Learning Containers 的 MXNet 在 Amazon ECS 上学习推理，请参阅Apache MXNet（孵化）推断。

PyTorch 训练

您必须先注册任务定义才能在 AmazECS 上运行任务。任务定义是分组在一起的一系列容器。以下示例使用了向Deep Learning Containers 添加训练脚本的 Docker 示例。

1. 使用以下内容创建名为 ecs-deep-learning-container-training-taskdef.json 的文件。

- 适用于 CPU

```
{
   "requiresCompatibilities":[
      "EC2"
   ],
   "containerDefinitions":[
      {
         "command":[
            "git clone https://github.com/pytorch/examples.git && python
 examples/mnist/main.py --no-cuda"
         ],
         "entryPoint":[
            "sh",
            "-c"
         ],
         "name":"pytorch-training-container",
         "image":"763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.5.1-cpu-py36-ubuntu16.04",
         "memory":4000,
         "cpu":256,
         "essential":true,
         "portMappings":[
            {
               "containerPort":80,
               "protocol":"tcp"
            }
         ],
         "logConfiguration":{
            "logDriver":"awslogs",
            "options":{
               "awslogs-group":"/ecs/pytorch-training-cpu",
               "awslogs-region":"us-east-1",
               "awslogs-stream-prefix":"mnist",
               "awslogs-create-group":"true"
            }
         }
      }
   ],
   "volumes":[

   ],
   "networkMode":"bridge",
```

```
   "placementConstraints":[

   ],
   "family":"pytorch"
}
```

- 对于 GPU

```
{
      "requiresCompatibilities": [
        "EC2"
      ],
      "containerDefinitions": [
        {
    "command": [
                "git clone https://github.com/pytorch/examples.git && python
 examples/mnist/main.py"
            ],
            "entryPoint": [
               "sh",
               "-c"
            ],
          "name": "pytorch-training-container",
          "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.5.1-gpu-py36-cu101-ubuntu16.04",
          "memory": 6111,
          "cpu": 256,
          "resourceRequirements" : [{
            "type" : "GPU",
            "value" : "1"
          }],
          "essential": true,
          "portMappings": [
            {
              "containerPort": 80,
              "protocol": "tcp"
            }
          ],
          "logConfiguration": {
              "logDriver": "awslogs",
              "options": {
                  "awslogs-group": "/ecs/pytorch-training-gpu",
                  "awslogs-region": "us-east-1",
                  "awslogs-stream-prefix": "mnist",
```

```
                    "awslogs-create-group": "true"
                }
            }
        }
    ],
    "volumes": [],
    "networkMode": "bridge",
    "placementConstraints": [],
    "family": "pytorch-training"
}
```

2. 注册任务定义。请记下输中的修订号，并在下这些中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-deep-learning-container-training-taskdef.json
```

3. 使用任务定义创建任务。您需要上中的修订标识符。

```
aws ecs run-task --cluster ecs-ec2-training-inference --task-definition pytorch:1
```

4. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 任务处于某种RUNNING状态后，选择任务标识符。
8. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

Amazon S3 插件 PyTorch

Deep Learning Containers 包含一个插件，使您可以使用来自 Amazon S3 ECECECTON 的数据进行 PyTorch 训练。

1. 要开始在 Amazon ECS 中使用 Amazon S3 插件，请使用您选择的区域设置您的AWS_REGION环境变量。

```
export AWS_REGION=us-east-1
```

2. 使用以下内容创建名为 ecs-deep-learning-container-pytorch-s3-plugin-taskdef.json 的文件。

- 适用于 CPU

```
{
   "requiresCompatibilities":[
      "EC2"
   ],
   "containerDefinitions":[
      {
         "command":[
            "git clone https://github.com/aws/amazon-s3-plugin-for-pytorch.git &&
 python amazon-s3-plugin-for-pytorch/examples/s3_imagenet_example.py"
         ],
         "entryPoint":[
            "sh",
            "-c"
         ],
         "name":"pytorch-s3-plugin-container",
         "image":"763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.8.1-cpu-py36-ubuntu18.04-v1.6",
         "memory":4000,
         "cpu":256,
         "essential":true,
         "portMappings":[
            {
               "containerPort":80,
               "protocol":"tcp"
            }
         ],
         "logConfiguration":{
            "logDriver":"awslogs",
            "options":{
               "awslogs-group":"/ecs/pytorch-s3-plugin-cpu",
               "awslogs-region":"us-east-1",
               "awslogs-stream-prefix":"imagenet",
               "awslogs-create-group":"true"
            }
         }
      }
   ],
   "volumes":[

   ],
   "networkMode":"bridge",
```

```
   "placementConstraints":[

   ],
   "family":"pytorch-s3-plugin"
}
```

- 对于 GPU

```
{
      "requiresCompatibilities": [
        "EC2"
      ],
      "containerDefinitions": [
        {
    "command": [
                "git clone https://github.com/aws/amazon-s3-plugin-
for-pytorch.git && python amazon-s3-plugin-for-pytorch/examples/
s3_imagenet_example.py"
             ],
             "entryPoint": [
                "sh",
                "-c"
             ],
          "name": "pytorch-s3-plugin-container",
          "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.8.1-gpu-py36-cu111-ubuntu18.04-v1.7",
          "memory": 6111,
          "cpu": 256,
          "resourceRequirements" : [{
            "type" : "GPU",
            "value" : "1"
          }],
          "essential": true,
          "portMappings": [
            {
              "containerPort": 80,
              "protocol": "tcp"
            }
          ],
          "logConfiguration": {
              "logDriver": "awslogs",
              "options": {
                  "awslogs-group": "/ecs/pytorch-s3-plugin-gpu",
                  "awslogs-region": "us-east-1",
```

```
                "awslogs-stream-prefix": "imagenet",
                    "awslogs-create-group": "true"
            }
          }
        }
      ],
      "volumes": [],
      "networkMode": "bridge",
      "placementConstraints": [],
      "family": "pytorch-s3-plugin"
    }
```

3. 注册任务定义。请记下输中的修订号，并在下这些中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-deep-learning-container-pytorch-s3-plugin-taskdef.json
```

4. 使用任务定义创建任务。您需要上中的修订标识符。

```
aws ecs run-task --cluster ecs-pytorch-s3-plugin --task-definition pytorch-s3-plugin:1
```

5. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
6. 选择 ecs-pytorch-s3-plugin 集群。
7. 在 Cluster 页面上，选择 Tasks。
8. 任务处于某种RUNNING状态后，选择任务标识符。
9. 在“日志”下，选择“查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看 Amazon S3 插件示例日志。

有关更多信息和其他示例，请参阅 PyTorch存储库的 Amazon S3 插件。

后续步骤

要使用D PyTorch eep Learning Containers 在 Amazon ECS 上学习推理，请参阅PyTorch 理推理推理。

推理

本节介绍如何使用 Apache MXNet（孵化）、、和 TensorFlow 2 在亚马逊弹性容器服务 (Amazon ECS) 的DeeAWS p Learning Containers 上运行推理。 PyTorch TensorFlow您还可以使用 Elastic

Inference 通过DeeAWS p Learning Containers 运行推理。有关Elastic Inference 的教程和更多信息，请参阅在 Amazon ECS 上使用带有Elastic Inference 的DeeAWS p Learning Container s。

有关Deep Learning Containers，请参阅Deep Learning Containers 映像。

> **Note**
>
> MKL 用户：阅读AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

> **Important**
>
> 如果您的账户已创建 Amamazon ECS 服务相关角色，则默认情况下会为您的服务使用该角色，除非在此处指定一个角色，则默认情况下会为您的服务使用该角色，除非在此处指定一个角色，则默认情况下会为您的服务使用该角色，如果您的任务定义使用 awsvpc 网络模式，则需要服务相关角色。如果将服务配置为使用服务发现、外部部署控制器、多个目标组或 Elastic Inference 加速器（在这种情况下，不应在此处指定角色），则需要使用该角色。有关更多信息，请参阅 Amazon ECS 开发人员指南中的 Amazon ECS 使用服务关联角色。

目录

TensorFlow理推理推理

以下示例使用示例 Docker 镜像，该镜像从主机的命令行向Deep Learning Containers 添加 CPU 或 GPU 推理脚本。

理推理推理推理推理推理推理

使用以下示例运行基于 CPU 的推理。

1. 使用以下内容创建名为 `ecs-dlc-cpu-inference-taskdef.json` 的文件。你可以将其与任一 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 映像更改为 TensorFlow 2 映像，然后克隆 r2.0 服务存储库分支而不是 r1.15。

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mkdir -p /test && cd /test && git clone -b r1.15 https://github.com/
tensorflow/serving.git && tensorflow_model_server --port=8500 --rest_api_port=8501
 --model_name=saved_model_half_plus_two --model_base_path=/test/serving/
tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_cpu"
  ],
  "entryPoint": [
   "sh",
   "-c"
  ],
  "name": "tensorflow-inference-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-cpu-py36-ubuntu18.04",
  "memory": 8111,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
    "hostPort": 8500,
    "protocol": "tcp",
    "containerPort": 8500
   },
   {
    "hostPort": 8501,
    "protocol": "tcp",
    "containerPort": 8501
   },
   {
    "containerPort": 80,
    "protocol": "tcp"
   }
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "/ecs/tensorflow-inference-gpu",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "half-plus-two",
    "awslogs-create-group": "true"
```

```
    }
   }
  }],
  "volumes": [],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "tensorflow-inference"
}
```

2. 注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中下一步中下一步中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-dlc-cpu-inference-taskdef.json
```

3. 创建 Amazon ECS 服务 指定任务定义时，`revision_id`替换为上一步输出中任务定义的修订版号。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                       --service-name cli-ec2-inference-cpu \
                       --task-definition Ec2TFInference:revision_id \
                       --desired-count 1 \
                       --launch-type EC2 \
                       --scheduling-strategy="REPLICA" \
                       --region us-east-1
```

4. 通过完成以下步骤来验证服务并获取网络终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 `ecs-ec2-training-inference` 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

   d. 任务处于`RUNNING`状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

   g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8501 的 IP 地址，并在下一步中使用。

5. 要运行推理，请使用以下命令。将上一步开始添加其一步开始添加其名称。

```
curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST http://<External ip>:8501/v1/
models/saved_model_half_plus_two:predict
```

下面是示例输出。

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

> **Important**
>
> 如果您无法连接到外部 IP 地址，请确保您的公司防火墙未阻塞非标准端口，如 8501。您可以尝试切换至来宾网络来验证。

基于 GPU 的推断

使用以下示例运行基于 GPU 的推理。

1. 使用以下内容创建名为 `ecs-dlc-gpu-inference-taskdef.json` 的文件。你可以将其与任一 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 映像更改为 TensorFlow 2 映像，然后克隆 r2.0 服务存储库分支而不是 r1.15。

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mkdir -p /test && cd /test && git clone -b r1.15 https://github.com/
tensorflow/serving.git && tensorflow_model_server --port=8500 --rest_api_port=8501
 --model_name=saved_model_half_plus_two --model_base_path=/test/serving/
tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_gpu"
  ],
  "entryPoint": [
   "sh",
   "-c"
  ],
  "name": "tensorflow-inference-container",
```

```
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-gpu-py36-cu100-ubuntu18.04",
  "memory": 8111,
  "cpu": 256,
  "resourceRequirements": [{
   "type": "GPU",
   "value": "1"
  }],
  "essential": true,
  "portMappings": [{
    "hostPort": 8500,
    "protocol": "tcp",
    "containerPort": 8500
   },
   {
    "hostPort": 8501,
    "protocol": "tcp",
    "containerPort": 8501
   },
   {
    "containerPort": 80,
    "protocol": "tcp"
   }
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "/ecs/TFInference",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs",
    "awslogs-create-group": "true"
   }
  }
 }],
 "volumes": [],
 "networkMode": "bridge",
 "placementConstraints": [],
 "family": "TensorFlowInference"
}
```

2. 注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中下一步中一步中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-dlc-gpu-inference-taskdef.json
```

3. 创建 Amazon ECS 服务 指定任务定义时，revision_id替换为上一步输出中任务定义的修订版号。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                       --service-name cli-ec2-inference-gpu \
                       --task-definition Ec2TFInference:revision_id \
                       --desired-count 1 \
                       --launch-type EC2 \
                       --scheduling-strategy="REPLICA" \
                       --region us-east-1
```

4. 通过完成以下步骤来验证服务并获取网络终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 ecs-ec2-training-inference 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

   d. 任务处于RUNNING状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

   g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8501 的 IP 地址，并在下一步中使用。

5. 要运行推理，请使用以下命令。将上一步开始添加其一步开始添加其名称。

```
curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST http://<External ip>:8501/v1/models/saved_model_half_plus_two:predict
```

   下面是示例输出。

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

> **Important**
>
> 如果您无法连接到外部 IP 地址，请确保您的公司防火墙未阻塞非标准端口，如 8501。您可以尝试切换至来宾网络来验证。

Apache MXNet（孵化）推断

您必须先注册任务定义，然后才能在 Amazon ECS 集群中运行任务定义，然后才能在 Amazon ECS 集群中运行任务定义，任务定义是分组在一起的一系列容器。以下示例使用示例 Docker 镜像，该镜像从主机的命令行向Deep Learning Containers 添加 CPU 或 GPU 推理脚本。

理推理推理推理推理推理

使用以下任务定义运行基于 CPU 的推理。

1. 使用以下内容创建名为 `ecs-dlc-cpu-inference-taskdef.json` 的文件。

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mxnet-model-server --start --mms-config /home/model-server/config.properties
 --models squeezenet=https://s3.amazonaws.com/model-server/models/squeezenet_v1.1/
squeezenet_v1.1.model"
  ],
  "name": "mxnet-inference-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.6.0-cpu-
py36-ubuntu16.04",
  "memory": 8111,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
    "hostPort": 8081,
    "protocol": "tcp",
    "containerPort": 8081
   },
   {
    "hostPort": 80,
```

```
    "protocol": "tcp",
    "containerPort": 8080
   }
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "/ecs/mxnet-inference-cpu",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "squeezenet",
    "awslogs-create-group": "true"
   }
  }
 }],
 "volumes": [],
 "networkMode": "bridge",
 "placementConstraints": [],
 "family": "mxnet-inference"
}
```

2. 注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中下一步中一步中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-dlc-cpu-inference-taskdef.json
```

3. 创建 Amazon ECS 服务 指定任务定义时，`revision_id`替换为上一步输出中任务定义的修订版号。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                       --service-name cli-ec2-inference-cpu \
                       --task-definition Ec2TFInference:revision_id \
                       --desired-count 1 \
                       --launch-type EC2 \
                       --scheduling-strategy REPLICA \
                       --region us-east-1
```

4. 验证服务并获取终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 `ecs-ec2-training-inference` 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

d. 任务处于RUNNING状态后，选择任务标识符。

e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

f. 在 Containers (容器) 下，展开容器详细信息。

g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8081 的 IP 地址，并在下一步中使用。

5. 要运行推理，请使用以下命令。将上external IP一步开始添加其一步开始添加其地址。

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl -X POST http://<External ip>/predictions/squeezenet -T kitten.jpg
```

下面是示例输出。

```
[
  {
    "probability": 0.8582226634025574,
    "class": "n02124075 Egyptian cat"
  },
  {
    "probability": 0.09160050004720688,
    "class": "n02123045 tabby, tabby cat"
  },
  {
    "probability": 0.037487514317035675,
    "class": "n02123159 tiger cat"
  },
  {
    "probability": 0.0061649843119084835,
    "class": "n02128385 leopard, Panthera pardus"
  },
  {
    "probability": 0.003171598305925727,
    "class": "n02127052 lynx, catamount"
  }
]
```

> **Important**
>
> 如果您无法连接到外部 IP 地址，请确保您的企业防火墙不会阻止非标准端口，如 8081。您可以尝试切换至来宾网络来验证。

基于 GPU 的推断

使用以下任务定义运行基于 GPU 的推理。

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mxnet-model-server --start --mms-config /home/model-server/config.properties
 --models  squeezenet=https://s3.amazonaws.com/model-server/models/squeezenet_v1.1/
squeezenet_v1.1.model"
  ],
  "name": "mxnet-inference-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.6.0-gpu-
py36-cu101-ubuntu16.04",
  "memory": 8111,
  "cpu": 256,
  "resourceRequirements": [{
   "type": "GPU",
   "value": "1"
  }],
  "essential": true,
  "portMappings": [{
    "hostPort": 8081,
    "protocol": "tcp",
    "containerPort": 8081
   },
   {
    "hostPort": 80,
    "protocol": "tcp",
    "containerPort": 8080
   }
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "/ecs/mxnet-inference-gpu",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "squeezenet",
    "awslogs-create-group": "true"
   }
  }
```

```
 }],
 "volumes": [],
 "networkMode": "bridge",
 "placementConstraints": [],
 "family": "mxnet-inference"
}
```

1. 使用以下命令注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中使用。

```
aws ecs register-task-definition --cli-input-json file://<Task definition file>
```

2. 要创建服务，请在以下命令中将 `revision_id` 替换为上一步中的输出。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                    --service-name cli-ec2-inference-gpu \
                    --task-definition Ec2TFInference:<revision_id> \
                    --desired-count 1 \
                    --launch-type "EC2" \
                    --scheduling-strategy REPLICA \
                    --region us-east-1
```

3. 验证服务并获取终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 `ecs-ec2-training-inference` 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

   d. 任务处于RUNNING状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

   g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8081 的 IP 地址，并在下一步中使用。

4. 要运行推理，请使用以下命令。将上`external IP`一步开始添加其一步开始添加其地址。

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl -X POST http://<External ip>/predictions/squeezenet -T kitten.jpg
```

   下面是示例输出。

```
[
  {
    "probability": 0.8582226634025574,
    "class": "n02124075 Egyptian cat"
  },
  {
    "probability": 0.09160050004720688,
    "class": "n02123045 tabby, tabby cat"
  },
  {
    "probability": 0.037487514317035675,
    "class": "n02123159 tiger cat"
  },
  {
    "probability": 0.0061649843119084835,
    "class": "n02128385 leopard, Panthera pardus"
  },
  {
    "probability": 0.003171598305925727,
    "class": "n02127052 lynx, catamount"
  }
]
```

Important

如果您无法连接到外部 IP 地址，请确保您的企业防火墙不会阻止非标准端口，如 8081。您可以尝试切换至来宾网络来验证。

PyTorch 理推理推理

您必须先注册任务定义，然后才能在 Amazon ECS 集群中运行任务定义，然后才能在 Amazon ECS 集群中运行任务定义，任务定义是分组在一起的一系列容器。以下示例使用了向Deep Learning Containers 添加 CPU 或 GPU 推理脚本的 Docker 示例。

理推理推理推理推理推理

使用以下任务定义运行基于 CPU 的推理。

1. 使用以下内容创建名为 `ecs-dlc-cpu-inference-taskdef.json` 的文件。

```
{
    "requiresCompatibilities": [
        "EC2"
    ],
    "containerDefinitions": [{
        "command": [
            "mxnet-model-server --start --mms-config /home/model-server/
config.properties --models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/
multi-model-server/densenet/densenet.mar"
        ],
        "name": "pytorch-inference-container",
        "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
inference:1.3.1-cpu-py36-ubuntu16.04",
        "memory": 8111,
        "cpu": 256,
        "essential": true,
        "portMappings": [{
                "hostPort": 8081,
                "protocol": "tcp",
                "containerPort": 8081
            },
            {
                "hostPort": 80,
                "protocol": "tcp",
                "containerPort": 8080
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/densenet-inference-cpu",
                "awslogs-region": "us-east-1",
                "awslogs-stream-prefix": "densenet",
                "awslogs-create-group": "true"
            }
        }
    }],
    "volumes": [],
    "networkMode": "bridge",
    "placementConstraints": [],
    "family": "pytorch-inference"
}
```

2. 注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中下一步中下一步中使用。

```
aws ecs register-task-definition --cli-input-json file://ecs-dlc-cpu-inference-taskdef.json
```

3. 创建 Amazon ECS 服务 指定任务定义时，`revision_id`替换为上一步输出中任务定义的修订版号。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                       --service-name cli-ec2-inference-cpu \
                       --task-definition Ec2PTInference:revision_id \
                       --desired-count 1 \
                       --launch-type EC2 \
                       --scheduling-strategy REPLICA \
                       --region us-east-1
```

4. 通过完成以下步骤来验证服务并获取网络终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 `ecs-ec2-training-inference` 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

   d. 任务处于某种`RUNNING`状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

   g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8081 的 IP 地址，并在下一步中使用。

5. 要运行推理，请使用以下命令。将上`external IP`一步开始添加其一步开始添加其地址。

```
curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://<External ip>/predictions/densenet -T flower.jpg
```

> **Important**
>
> 如果您无法连接到外部 IP 地址，请确保您的企业防火墙不会阻止非标准端口，如 8081。您可以尝试切换至来宾网络来验证。

基于 GPU 的推断

使用以下任务定义运行基于 GPU 的推理。

```
{
    "requiresCompatibilities": [
        "EC2"
    ],
    "containerDefinitions": [{
        "command": [
            "mxnet-model-server --start --mms-config /home/model-server/
config.properties --models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/multi-
model-server/densenet/densenet.mar"
        ],
        "name": "pytorch-inference-container",
        "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.3.1-
gpu-py36-cu101-ubuntu16.04",
        "memory": 8111,
        "cpu": 256,
        "essential": true,
        "portMappings": [{
                "hostPort": 8081,
                "protocol": "tcp",
                "containerPort": 8081
            },
            {
                "hostPort": 80,
                "protocol": "tcp",
                "containerPort": 8080
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/densenet-inference-cpu",
                "awslogs-region": "us-east-1",
                "awslogs-stream-prefix": "densenet",
                "awslogs-create-group": "true"
            }
        }
    }],
    "volumes": [],
    "networkMode": "bridge",
    "placementConstraints": [],
```

```
    "family": "pytorch-inference"
}
```

1. 使用以下命令注册任务定义。请记下输出的内容，以便在下一步中下一步中下一步中下一步中下一步中使用。

```
aws ecs register-task-definition --cli-input-json file://<Task definition file>
```

2. 要创建服务，请在以下命令中将 revision_id 替换为上一步中的输出。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
                    --service-name cli-ec2-inference-gpu \
                    --task-definition Ec2PTInference:<revision_id> \
                    --desired-count 1 \
                    --launch-type "EC2" \
                    --scheduling-strategy REPLICA \
                    --region us-east-1
```

3. 通过完成以下步骤来验证服务并获取网络终端节点。

   a. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

   b. 选择 ecs-ec2-training-inference 集群。

   c. 在 Cluster (集群) 页面上，选择 Services (服务)，然后选择 cli-ec2-inference-cpu。

   d. 任务处于RUNNING状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看日志” CloudWatch。这将带您进入 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

   g. 在 “名称” 和 “网络绑定” 下，在 “外部链接” 下记下端口 8081 的 IP 地址，并在下一步中使用。

4. 要运行推理，请使用以下命令。将上external IP一步开始添加其一步开始添加其地址。

```
curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://<External ip>/predictions/densenet -T flower.jpg
```

> **Important**
>
> 如果您无法连接到外部 IP 地址，请确保您的企业防火墙不会阻止非标准端口，如 8081。您可以尝试切换至来宾网络来验证。

后续步骤

要了解如何在 Amazon ECS 上将自定义入口点与Deep Learning Containers 一起使用，请参阅自定义入口点。

自定义入口点

对于某些映像，Deep Learning Containers 使用自定义入口点脚本。如果您要使用自己的入口点，可以按如下方式覆盖入口点。

修改entryPoint包含任务定义的 JSON 文件中的参数。将文件路径包含到自定义入口点脚本中。下面说明这一例子。

```
"entryPoint":[
          "sh",
          "-c",
          "/usr/local/bin/mxnet-model-server --start --foreground --mms-config /home/
model-server/config.properties --models densenet=https://dlc-samples.s3.amazonaws.com/
pytorch/multi-model-server/densenet/densenet.mar"],
```

Amazon EKS

Amazon EKS 教程提供训练和推理示例，并演示如何设置和使用AWSDeep Learning Containers eee

- Amazon Elastic Kubernetes Service (Amazon EKS)
- Kubeflow 开启AWS

Kubeflow 开启AWS是 Kubeflow 的优化开源版本，适用于Amazon Elastic Kubernetes Service (Amazon EKS)。有关更多信息，请参阅AWSKubefloow 的功能.

> **Note**
>
> 中的所有训练和推理示例本部分在单节点集群上运行。

在运行任何示例之前，请访问Amazon EK 设置要么AWS安装时的 Kubeflow并按照安装说明部署 Amazon EKS 集群。

Kubeflow 的安装说明开启AWS在部署 Amazon EKS 集群之前，请先了解创建 AmazonAWSKubeflow 的分发。

目录

Amazon EK 设置

本节提供安装说明，以设置正在运行的深度学习环境AWSAmazon Elastic Kubernetes Service (Amazon EKS) 上的Deep Learning Containers

自定义映像

如果您要加载自己的代码或数据集并让其在您集群的每个节点均可用，则自定义映像将很有用。提供了使用自定义镜像的示例。你可以试用它们来开始使用，而无需创建自己的版本。

- 构建AWSDeep Learning Containers 自定义映像

许可

要使用 GPU 硬件，请使用具有必要 GPU 驱动程序的亚马逊机器映像。我们建议使用支持 GPU 的 Amazon EKS 优化的 AMI，本指南的后续步骤中将使用该功能。此 AMI 包含的软件不是AWS，因此需要最终用户许可协议 (EULA)。您必须在中订阅经过 EKS 优化的 AMIAWS Marketplace并接受 EULA，然后才能在工作节点组中使用 AMI。

> **Important**
>
> 要订阅 AMI，请访问AWSMarketplace.

配置安全设置

要使用 Amazon EKS，您必须拥有一个可以访问多个安全权限的用户账户。这些设置为AWS Identity and Access Management(IAM) 工具。

1. 按照中的步骤创建 IAM 用户或更新现有 IAM 用户在您的中创建 IAM 用户AWS帐户.
2. 获取此用户的凭证。

a. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。

b. 下面Users，请选择您的用户。

c. Select Security Credentials.

d. Select Create access key.

e. 下载key pair 或复制信息以供日后使用。

3. 向您的 IAM 用户添加以下策略。这些策略为Amazon EKS、IAmazon EKS、Amazon EKS、AElastic Compute Cloud (Amazon EC2) 提供了所需的访问权限。

a. Select Permissions.

b. Select Add permissions.

c. Select Create policy.

d. 来自的Create policy窗口，选择JSON选项卡。

e. 粘贴以下内容。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

f. 给策略命名EKSFullAccess并创建策略。

g. 导航回到Grant permissions时段。

h. Select Attach existing policies directly.

i. 搜索EKSFullAccess，然后选中该复选框。

j. 搜索AWSCloudFormationFullAccess，然后选中该复选框。

k. 搜索AmazonEC2FullAccess，然后选中该复选框。

l. 搜索IAMFullAccess，然后选中该复选框。

m. 搜索AmazonEC2ContainerRegistryReadOnly，然后选中该复选框。

n. 搜索AmazonEKS_CNI_Policy，然后选中该复选框。

o. 搜索AmazonS3FullAccess，然后选中该复选框。

p. 接受更改。

网关节点

要设置 Amazon EKS 集群，请使用开源工具eksctl. 我们建议您使用带有深度学习基础 AMI (Ubuntu) 的 Amazon EC2 实例来分配和控制您的集群。您可以在您的计算机或已经运行的 Amazon EC2 实例上本地运行这些工具。但是，为了简化本指南，我们假设您使用的是带有 Ubuntu 16.04 的深度学习基础 AMI (DLAMI)。我们将此称为您的网关节点。

在开始之前，请考虑您的训练数据的位置或您要运行集群以响应推理请求的位置。通常情况下，您的用于训练或推理的数据和集群应位于同一区域。此外，您在同一个区域中启动您的网关节点。你可以快点关注这个10 分钟教程它指导您启动一个 DLAMI 以用作您的网关节点。

1. 登录到您的网关节点。
2. 安装或升级AWSCLI。要访问所需的新 Kubernetes 功能，您必须具有最新版本。

```
$ sudo pip install --upgrade awscli
```

3. 安装eksctl通过在中运行与您的操作系统对应的命令Amazon EKS 用户指南的安装说明。有关更多信息eksctl，另请参阅eksctl 文档.
4. 安装kubectl按中的步骤操作正在安装 kubectl指南。

> **Note**
>
> 您必须使用kubectl与您的 Amazon EKS 集群控制层面版本不同的一个次要版本内的版本。例如，一个 1.18kubectl客户端可以使用 Kubernetes 1.17、1.18 和 1.19 集群。

5. 通过运行以下命令安装 aws-iam-authenticator。有关更多信息 aws-iam-authenticator，请参阅正在安装aws-iam-authenticator.

```
$ curl -o aws-iam-authenticator https://amazon-eks.s3.us-
west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/aws-iam-authenticator
$ chmod +x aws-iam-authenticator
$ cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$HOME/
bin:$PATH
```

6. 从“Security Configuration (安全配置)”部分中运行 IAM 用户的 `aws configure`。您正在复制 IAM 用户的AWS访问密钥，然后是AWS您在 IAM 控制台中访问的私有访问密钥并将其粘贴到来自的提示中`aws configure`.

GPU 集群

1. 检查以下使用 p3.8xlarge 实例类型创建集群的命令。在运行它之前，必须进行以下修改。

   - `name`是你用来管理集群的东西。您可以将 `cluster-name` 更改为希望的任何名称，只要其中没有空格或特殊字符。
   - `eks-version`是Amazon EKSkubernetes 版本。有关支持的亚马逊 EKS 版本，请参阅可用的 Amazon EKS Kubernetes 版本.
   - `nodes`是您要在集群中存储的实例的数量。在本示例中，我们将从三个节点开始。
   - `node-type`指的是实例类.
   - `timeout`和`*ssh-access *`可以独自一人。
   - `ssh-public-key`是要用于登录工作节点的密钥的名称。要么使用你已经使用的安全密钥，要么创建一个新的安全密钥，但一定要换掉 ssh-public-key 使用为你使用的地区分配的密钥。注意：您只需要提供密钥名称，如 Amazon EC2 控制台的 “密钥对” 部分所示。
   - `region`是启动集群的 Amazon EC2 区域。如果您计划使用位于特定区域（除外）的训练数据*<us-east-1>*）我们建议您使用相同的区域。这些区域有： ssh-public-key 必须有权在该区域启动实例。

   > **Note**
   >
   > 本指南的其余部分假定区域是 *<us-east-1>*。

2. 更改命令后，运行该命令并等待。单节点集群可能需要几分钟，如果您选择创建大型集群，则可能需要更长的时间。

```
$ eksctl create cluster <cluster-name> \
                      --version <eks-version> \
                      --nodes 3 \
                      --node-type=<p3.8xlarge> \
                      --timeout=40m \
                      --ssh-access \
                      --ssh-public-key <key_pair_name> \
                      --region <us-east-1> \
```

```
                    --zones=us-east-1a,us-east-1b,us-east-1d \
                    --auto-kubeconfig
```

您应该可以看到类似于如下输出的内容：

```
EKS cluster "training-1" in "us-east-1" region is ready
```

3. 理想情况下，auto-kubeconfig 应已配置您的集群。但是，如果您遇到问题，则可以运行以下命令来设置您的 kubeconfig。如果您要从其他位置更改网关节点和管理集群，也可使用此命令。

```
$ aws eks --region <region> update-kubeconfig --name <cluster-name>
```

您应该可以看到类似于如下输出的内容：

```
Added new context arn:aws:eks:us-east-1:999999999999:cluster/training-1 to /home/
ubuntu/.kube/config
```

4. 如果您计划使用 GPU 实例类型，请务必运行适用于 Kubernetes 的 NVIDIA 设备插件在集群中使用以下命令：

```
$ kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/
v1.12/nvidia-device-plugin.yml
$ kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/
v0.9.0/nvidia-device-plugin.yml
```

5. 验证 GPU 在您集群的每个节点上是否可用

```
$ kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

CPU 集群

请参阅上一节中有关使用方法的讨论eksctl命令启动 GPU 集群，然后修改node-type使用 CPU 实例类型。

哈瓦那集群

请参阅前面关于使用方法的讨论eksctl命令启动 GPU 集群，然后修改node-type使用带有 Habana Gaudi 加速器的实例，例如DL1 实例类型。

测试您的集群

1. 你可以运行一个kubectl命令在集群上检查其状态。试用该命令以确保其选择的是您要管理的当前集群。

```
$ kubectl get nodes -o wide
```

2. 简单了解 ~/.kube。此目录具有用于从您的网关节点配置的各个集群的 kubeconfig 文件。如果进一步浏览到该文件夹，您可以找到 ~/.kube/eksctl/clusters - 此文件夹包含用于使用 eksctl 创建的集群的 kubeconfig 文件。该文件包含一些理想情况下不必修改的细节，因为这些工具正在为您生成和更新配置，但是在进行故障排除时最好参考。

3. 验证集群是否处于活动状态。

```
$ aws eks --region <region> describe-cluster --name <cluster-name> --query
 cluster.status
```

您应看到以下输出：

```
"ACTIVE"
```

4. 如果您在同一主机实例中具有多个集群设置，请验证 kubectl 上下文。有时候，确保找到的默认上下文会有所帮助kubectl设置正确。使用以下命令检查此内容：

```
$ kubectl config get-contexts
```

5. 如果未按预期设置该上下文，请使用以下命令修复此问题：

```
$ aws eks --region <region> update-kubeconfig --name <cluster-name>
```

管理您的集群

当你想控制或查询集群时，你可以使用 kubeconfig 参数通过配置文件对其进行寻址。这在您有多个集群时很有用。例如，如果你有一个名为 “training-gpu-1” 的单独集群，你可以调用get pods通过将配置文件作为参数传递来对其执行命令，如下所示：

```
$ kubectl --kubeconfig=/home/ubuntu/.kube/eksctl/clusters/training-gpu-1 get pods
```

值得注意的是，你可以在没有 kubeconfig 参数的情况下运行同样的命令。在这种情况下，该命令将使用当前主动控制的集群 (current-context)。

```
$ kubectl get pods
```

如果您设置了多个集群，而这些集群尚未安装 NVIDIA 插件，则可以采用以下方式安装该插件：

```
$ kubectl --kubeconfig=/home/ubuntu/.kube/eksctl/clusters/training-gpu-1 create -f
 https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-
plugin.yml
```

您还可以通过更新 kubeconfig、传递要管理的集群的名称来更改活动集群。以下命令更新 kubeconfig 且无需使用 kubeconfig 参数。

```
$ aws eks –region us-east-1 update-kubeconfig –name training-gpu-1
```

如果您遵循本指南中的所有示例，则可能会经常在活动集群之间切换。这样你就可以协调训练或推理，或者使用在不同集群上运行的不同框架。

清除

使用完集群后，将其删除，以避免产生额外费用。

```
$ eksctl delete cluster --name=<cluster-name>
```

要仅删除 Pod，运行以下命令：

```
$ kubectl delete pods <name>
```

要重置访问集群的密钥，请运行以下命令：

```
$ kubectl delete secret ${SECRET} -n ${NAMESPACE} || true
```

删除nodegroup连接到集群，运行以下命令：

```
$ eksctl delete nodegroup --name <cluster_name>
```

要附加一个nodegroup到集群，运行以下命令：

```
$ eksctl create nodegroup
```

```
                    --cluster <cluster-name> \
                    --node-ami <ami_id> \
                    --nodes <num_nodes> \
                    --node-type=<instance_type> \
                    --timeout=40m \
                    --ssh-access \
                    --ssh-public-key <key_pair_name> \
                    --region <us-east-1> \
                    --auto-kubeconfig
```

后续步骤

要了解如何在 Amazon EKS 上使用Deep Learning Containers 进行训练和推理，请访问培训要么Inference.

目录

培训

使用中的步骤创建集群后Amazon EK 设置，你可以用它来运行训练作业。对于训练，您可以使用 CPU、GPU 或分布式 GPU 示例，具体取决于集群中的节点。以下部分中的主题介绍如何使用 Apache MXNet（孵化），PyTorch, TensorFlow，以及 TensorFlow 2 个训练示例。

目录

CPU 训练

本部分用于基于 CPU 的容器的培训。

有关DeDeep Learning Containers arning Containers 的完整列表，请参阅Deep Learning Containers 映像. 有关使用英特尔数学内核库 (MKL) 时的最佳配置设置的提示，请参阅AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议.

目录

Apache MXNet（孵化中）CPU 训练

本教程指导您在单节点 CPU 集群上使用 Apache MXNet（孵化）进行训练。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将下载 MXNet 存储库并运行 MNIST 示例。Open（打开）vi要么vim并复制并粘贴以下内容。将此文件另存为 `mxnet.yaml`。

```
apiVersion: v1
kind: Pod
metadata:
  name: mxnet-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: mxnet-training
    image:  763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.6.0-cpu-py36-ubuntu16.04
    command: ["/bin/sh","-c"]
    args: ["git clone -b v1.4.x https://github.com/apache/incubator-mxnet.git &&
 python ./incubator-mxnet/example/image-classification/train_mnist.py"]
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f mxnet.yaml
```

3. 您应看到以下输出：

```
pod/mxnet-training created
```

4. 检查状态。任务“mxnet-training”的名称位于 mxnet.yaml 文件中。它现在将显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
mxnet-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs mxnet-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'incubator-mxnet'...
INFO:root:Epoch[0] Batch [0-100]    Speed: 18437.78 samples/sec
 accuracy=0.777228
INFO:root:Epoch[0] Batch [100-200]    Speed: 16814.68 samples/sec
 accuracy=0.907188
INFO:root:Epoch[0] Batch [200-300]    Speed: 18855.48 samples/sec
 accuracy=0.926719
INFO:root:Epoch[0] Batch [300-400]    Speed: 20260.84 samples/sec
 accuracy=0.938438
INFO:root:Epoch[0] Batch [400-500]    Speed: 9062.62 samples/sec
 accuracy=0.938594
INFO:root:Epoch[0] Batch [500-600]    Speed: 10467.17 samples/sec
 accuracy=0.945000
INFO:root:Epoch[0] Batch [600-700]    Speed: 11082.03 samples/sec
 accuracy=0.954219
INFO:root:Epoch[0] Batch [700-800]    Speed: 11505.02 samples/sec
 accuracy=0.956875
INFO:root:Epoch[0] Batch [800-900]    Speed: 9072.26 samples/sec
 accuracy=0.955781
INFO:root:Epoch[0] Train-accuracy=0.923424
...
```

6. 查看日志以查看训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时Completed”，培训工作已经完成。

后续步骤

要在 Amazon EKS 上使用带有Deep Learning Containers 的 MXNet 学习基于 CPU 的推理，请参阅Apache MxNet（孵化）CPU 推理.

TensorFlow CPU 训练

本教程指导你进行训练 TensorFlow 您的单节点 CPU 集群上的模型。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。Open（打开）vi要么vim并复制并粘贴以下内容。将此文件另存为 `tf.yaml`。您可以将其与任一使用 TensorFlow要么 TensorFlow 2. 要与之配合使用 TensorFlow 2，将 Docker 镜像更改为 TensorFlow 2 张图片。

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: tensorflow-training
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.2-cpu-py36-ubuntu18.04
    command: ["/bin/sh","-c"]
    args: ["git clone https://github.com/fchollet/keras.git && python /keras/
examples/mnist_cnn.py"]
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f tf.yaml
```

3. 您应看到以下输出：

```
pod/tensorflow-training created
```

4. 检查状态。任务“tensorflow-training”的名称位于 tf.yaml 文件中。它现在将显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
tensorflow-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs tensorflow-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'keras'...
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz

    8192/11490434 [..............................] - ETA: 0s
 6479872/11490434 [===============>..............] - ETA: 0s
 8740864/11490434 [=====================>........] - ETA: 0s
11493376/11490434 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
2019-03-19 01:52:33.863598: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
 use: AVX512F
2019-03-19 01:52:33.867616: I tensorflow/core/common_runtime/process_util.cc:69]
 Creating new thread pool with default inter op setting: 2. Tune using
 inter_op_parallelism_threads for best performance.

  128/60000 [..............................] - ETA: 10:43 - loss: 2.3076 - acc:
 0.0625
  256/60000 [..............................] - ETA: 5:59 - loss: 2.2528 - acc:
 0.1445
  384/60000 [..............................] - ETA: 4:24 - loss: 2.2183 - acc:
 0.1875
  512/60000 [..............................] - ETA: 3:35 - loss: 2.1652 - acc:
 0.1953
  640/60000 [..............................] - ETA: 3:05 - loss: 2.1078 - acc:
 0.2422
```

```
...
```

6. 您可以检查日志以观察训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时Completed” 你会知道培训工作已经完成。

后续步骤

要在 Amazon EKS 上学习基于 CPU 的推理，请使用 TensorFlow 有关DDeep Learning Containers ContaTensorFlow CPU 推理.

PyTorch CPU 训练

本教程将指导您完成训练 PyTorch 在你的单节点 CPU pod 上建模。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。这个 pod 文件将下载 PyTorch存储库并运行 MNIST 示例。Open（打开）vi要么vim，然后复制并粘贴以下内容。将此文件另存为 pytorch.yaml。

```
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-training
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.5.1-cpu-
py36-ubuntu16.04
    command:
      - "/bin/sh"
      - "-c"
    args:
    - "git clone https://github.com/pytorch/examples.git && python examples/mnist/
main.py --no-cuda"
    env:
    - name: OMP_NUM_THREADS
      value: "36"
    - name: KMP_AFFINITY
      value: "granularity=fine,verbose,compact,1,0"
    - name: KMP_BLOCKTIME
      value: "1"
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f pytorch.yaml
```

3. 您应看到以下输出：

```
pod/pytorch-training created
```

4. 检查状态。作业“pytorch-training”的名称位于 pytorch.yaml 文件中。它现在将显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
pytorch-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs pytorch-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'examples'...
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/
MNIST/raw/train-images-idx3-ubyte.gz
9920512it [00:00, 40133996.38it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/
MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
32768it [00:00, 831315.84it/s]
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/
MNIST/raw/t10k-images-idx3-ubyte.gz
1654784it [00:00, 13019129.43it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/
MNIST/raw/t10k-labels-idx1-ubyte.gz
8192it [00:00, 337197.38it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
```

```
Processing...
Done!
Train Epoch: 1 [0/60000 (0%)]    Loss: 2.300039
Train Epoch: 1 [640/60000 (1%)]    Loss: 2.213470
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.170460
Train Epoch: 1 [1920/60000 (3%)]    Loss: 2.076699
Train Epoch: 1 [2560/60000 (4%)]    Loss: 1.868078
Train Epoch: 1 [3200/60000 (5%)]    Loss: 1.414199
Train Epoch: 1 [3840/60000 (6%)]    Loss: 1.000870
```

6. 查看日志以查看训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时Completed” 你会知道培训工作已经完成。

请参阅清除了解有关在使用完集群后清理集群的信息。

Amazon S3 插件适用于 PyTorch

Deep Learning Containers 包含一个插件，该插件使您可以将来自 Amazon S3 桶的数据用于 PyTorch 训练。

1. 要开始在 Amazon EKS 上使用 Amazon S3 插件，请检查以确保您的集群实例具有对 Amazon S3 的完全访问权限。创建 IAM 角色它授予 Amazon S3 访问Amazon EC2 实例的权限，并将角色附加到您的实例。您可以使用亚马逊 S3FullAccess要么亚马逊 S3ReadOnlyAccess策略。
2. 设置您的AWS_REGION带有您选择的区域的环境变量。

```
export AWS_REGION=us-east-1
```

3. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将使用 PyTorch Amazon S3 插件用于访问示例 Amazon S3 数据集。

> **Note**
>
> 你的 CPU 集群应该使用c5.12xlarge在本示例中，节点或更大。

Open（打开）vi要么vim，然后复制并粘贴以下内容。将此文件另存为 s3plugin.yaml。

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: pytorch-s3-plugin
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-s3-plugin
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.8.1-cpu-
py36-ubuntu18.04-v1.6
    command:
      - "/bin/sh"
      - "-c"
    args:
    - "git clone https://github.com/aws/amazon-s3-plugin-for-pytorch.git && python
 amazon-s3-plugin-for-pytorch/examples/s3_imagenet_example.py"
    env:
    - name: OMP_NUM_THREADS
      value: "36"
    - name: KMP_AFFINITY
      value: "granularity=fine,verbose,compact,1,0"
    - name: KMP_BLOCKTIME
      value: "1"
```

4. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f s3plugin.yaml
```

5. 检查状态。任务名称pytorch-s3-plugin这是在中指定的s3plugin.yaml文件现在将出现在状态信息旁边。您可以多次运行以下命令，直到看到状态更改为”Running。”

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
pytorch-s3-plugin 0/1 Running 8 19m
```

6. 查看日志以了解更多详细信息。

```
$ kubectl logs pytorch-s3-plugin
```

有关更多信息，请参阅 。Amazon S3 插件适用于 PyTorch存储库。

后续步骤

要在 Amazon EKS 上学习基于 CPU 的推理，请使用 PyTorch 有关DDeep Learning Containers ContaPyTorch CPU 推理.

GPU 训练

本部分用于在基于 GPU 的集群上进行培训。

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像. 有关使用英特尔数学内核库 (MKL) 时的最佳配置设置的提示，请参阅AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议.

目录

Apache MXNet（孵化中）GPU 训练

本教程指导您在单节点 GPU 集群上使用 Apache MXNet（孵化）进行训练。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将下载 MXNet 存储库并运行 MNIST 示例。Open（打开）vi要么vim并复制并粘贴以下内容。将此文件另存为 `mxnet.yaml`。

```
apiVersion: v1
kind: Pod
metadata:
  name: mxnet-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: mxnet-training
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-training:1.6.0-gpu-
py36-cu101-ubuntu16.04
    command: ["/bin/sh","-c"]
    args: ["git clone -b v1.4.x https://github.com/apache/incubator-mxnet.git &&
 python ./incubator-mxnet/example/image-classification/train_mnist.py"]
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f mxnet.yaml
```

3. 您应看到以下输出：

```
pod/mxnet-training created
```

4. 检查状态。任务“tensorflow-training”的名称位于 tf.yaml 文件中。它现在将显示在状态中。如果您正在运行任何其他测试或之前已运行某些内容，它将显示在此列表中。多次运行，直到看到状态更改为”Running”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
mxnet-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs mxnet-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'incubator-mxnet'...
INFO:root:Epoch[0] Batch [0-100]    Speed: 18437.78 samples/sec
 accuracy=0.777228
INFO:root:Epoch[0] Batch [100-200]    Speed: 16814.68 samples/sec
 accuracy=0.907188
INFO:root:Epoch[0] Batch [200-300]    Speed: 18855.48 samples/sec
 accuracy=0.926719
INFO:root:Epoch[0] Batch [300-400]    Speed: 20260.84 samples/sec
 accuracy=0.938438
INFO:root:Epoch[0] Batch [400-500]    Speed: 9062.62 samples/sec
 accuracy=0.938594
INFO:root:Epoch[0] Batch [500-600]    Speed: 10467.17 samples/sec
 accuracy=0.945000
INFO:root:Epoch[0] Batch [600-700]    Speed: 11082.03 samples/sec
 accuracy=0.954219
```

```
INFO:root:Epoch[0] Batch [700-800]    Speed: 11505.02 samples/sec
 accuracy=0.956875
INFO:root:Epoch[0] Batch [800-900]    Speed: 9072.26 samples/sec
 accuracy=0.955781
INFO:root:Epoch[0] Train-accuracy=0.923424
...
```

6. 查看日志以查看训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时Completed”，培训工作已经完成。

后续步骤

要在 Amazon EKS 上使用带有Deep Learning Containers 的 MXNet 学习基于 GPU 的推理，请参阅Apache MxNet（孵化）GPU 推理.

TensorFlow GU 训练

本教程指导你进行训练 TensorFlow 您的单节点 GPU 集群上的模型。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。Open（打开）vi要么vim并复制并粘贴以下内容。将此文件另存为 `tf.yaml`。您可以将其与任一使用 TensorFlow要么 TensorFlow 2. 要与之配合使用 TensorFlow 2，将 Docker 镜像更改为 TensorFlow 2 张图片。

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: tensorflow-training
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:1.15.2-
gpu-py37-cu100-ubuntu18.04
    command: ["/bin/sh","-c"]
    args: ["git clone https://github.com/fchollet/keras.git && python /keras/
examples/mnist_cnn.py"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f tf.yaml
```

3. 您应看到以下输出：

```
pod/tensorflow-training created
```

4. 检查状态。任务“tensorflow-training”的名称位于 tf.yaml 文件中。它现在将显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行，直到看到状态更改为”Running”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
tensorflow-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs tensorflow-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'keras'...
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz

    8192/11490434 [..............................] - ETA: 0s
 6479872/11490434 [===============>..............] - ETA: 0s
 8740864/11490434 [=====================>........] - ETA: 0s
11493376/11490434 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
2019-03-19 01:52:33.863598: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
 use: AVX512F
```

```
2019-03-19 01:52:33.867616: I tensorflow/core/common_runtime/process_util.cc:69]
 Creating new thread pool with default inter op setting: 2. Tune using
 inter_op_parallelism_threads for best performance.

  128/60000 [..............................] - ETA: 10:43 - loss: 2.3076 - acc:
 0.0625
  256/60000 [..............................] - ETA: 5:59 - loss: 2.2528 - acc:
 0.1445
  384/60000 [..............................] - ETA: 4:24 - loss: 2.2183 - acc:
 0.1875
  512/60000 [..............................] - ETA: 3:35 - loss: 2.1652 - acc:
 0.1953
  640/60000 [..............................] - ETA: 3:05 - loss: 2.1078 - acc:
 0.2422
   ...
```

6. 查看日志以查看训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时Completed”，培训工作已经完成。

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，请使用 TensorFlow 有关DDeep Learning Containers ContaTensorFlow GPU 推理.

PyTorch GU 训练

本教程指导您使用以下方法进行训练 PyTorch 在你的单节点 GPU 集群上。

1. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。这个 pod 文件将下载 PyTorch存储库并运行 MNIST 示例。Open（打开）vi要么vim，然后复制并粘贴以下内容。将此文件另存为 pytorch.yaml。

```
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-training
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-training
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.5.1-gpu-
py36-cu101-ubuntu16.04
    command:
```

```
      - "/bin/sh"
      - "-c"
    args:
    - "git clone https://github.com/pytorch/examples.git && python examples/mnist/
main.py --no-cuda"
    env:
    - name: OMP_NUM_THREADS
      value: "36"
    - name: KMP_AFFINITY
      value: "granularity=fine,verbose,compact,1,0"
    - name: KMP_BLOCKTIME
      value: "1"
```

2. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f pytorch.yaml
```

3. 您应看到以下输出：

```
pod/pytorch-training created
```

4. 检查状态。作业“pytorch-training”的名称位于 pytorch.yaml 文件中。它现在将显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行，直到看到状态更改为”Running”。

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
pytorch-training 0/1 Running 8 19m
```

5. 检查日志以查看训练输出。

```
$ kubectl logs pytorch-training
```

您应该可以看到类似于如下输出的内容：

```
Cloning into 'examples'...
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/
MNIST/raw/train-images-idx3-ubyte.gz
```

```
9920512it [00:00, 40133996.38it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/
MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
32768it [00:00, 831315.84it/s]
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/
MNIST/raw/t10k-images-idx3-ubyte.gz
1654784it [00:00, 13019129.43it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/
MNIST/raw/t10k-labels-idx1-ubyte.gz
8192it [00:00, 337197.38it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
Processing...
Done!
Train Epoch: 1 [0/60000 (0%)]    Loss: 2.300039
Train Epoch: 1 [640/60000 (1%)]    Loss: 2.213470
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.170460
Train Epoch: 1 [1920/60000 (3%)]    Loss: 2.076699
Train Epoch: 1 [2560/60000 (4%)]    Loss: 1.868078
Train Epoch: 1 [3200/60000 (5%)]    Loss: 1.414199
Train Epoch: 1 [3840/60000 (6%)]    Loss: 1.000870
```

6. 查看日志以查看训练进度。你也可以继续查看”get pods” 刷新状态。当状态更改为 “” 时`Completed`“，培训工作已经完成。

请参阅清除了解有关在使用完集群后清理集群的信息。

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，请使用 PyTorch 有关DDeep Learning Containers ContaPyTorch GPU 推理.

Amazon S3 插件适用于 PyTorch

Deep Learning Containers 包含一个插件，该插件使您可以将来自 Amazon S3 桶的数据用于 PyTorch 训练。

1. 要开始在 Amazon EKS 上使用 Amazon S3 插件，请检查以确保您的集群实例具有对 Amazon S3 的完全访问权限。创建 IAM 角色它授予 Amazon S3 访问Amazon EC2 实例的权限，并将角色附加到您的实例。您可以使用亚马逊 S3FullAccess要么亚马逊 S3ReadOnlyAccess策略。

2. 设置您的AWS_REGION带有您选择的区域的环境变量。

```
export AWS_REGION=us-east-1
```

3. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行什么的说明。此 pod 文件将使用 PyTorch Amazon S3 插件用于访问示例 Amazon S3 数据集。

> **Note**
>
> 你的 GPU 集群应该使用p3.8xlarge在本示例中，节点或更大。

Open（打开）vi要么vim，然后复制并粘贴以下内容。将此文件另存为 s3plugin.yaml。

```
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-s3-plugin
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-s3-plugin
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.8.1-gpu-
py36-cu111-ubuntu18.04-v1.7
    command:
      - "/bin/sh"
      - "-c"
    args:
    - "git clone https://github.com/aws/amazon-s3-plugin-for-pytorch.git && python
 amazon-s3-plugin-for-pytorch/examples/s3_imagenet_example.py"
    env:
    - name: OMP_NUM_THREADS
      value: "36"
    - name: KMP_AFFINITY
      value: "granularity=fine,verbose,compact,1,0"
    - name: KMP_BLOCKTIME
      value: "1"
```

4. 使用以下命令将 pod 文件分配给集群kubectl.

```
$ kubectl create -f s3plugin.yaml
```

5. 检查状态。任务名称pytorch-s3-plugin这是在中指定的s3plugin.yaml文件现在将出现在状态信息旁边。您可以多次运行以下命令，直到看到状态更改为”Running。”

```
$ kubectl get pods
```

您应看到以下输出：

```
NAME READY STATUS RESTARTS AGE
pytorch-s3-plugin 0/1 Running 8 19m
```

6. 查看日志以了解更多详细信息。

```
$ kubectl logs pytorch-s3-plugin
```

有关更多信息，请参阅 。Amazon S3 插件适用于 PyTorch存储库。

分布式 GPU 训练

本部分针对在多节点 GPU 集群上运行分布式训练。

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像.

目录

设置集群以进行分布式训练

要在 EKS 上运行分布式训练，您需要在集群上安装以下组件。

- 默认安装的Kubeflow使用必需的组件，例如 PyTorch 运算符， TensorFlow 操作员和 NVIDIA 插件。
- Apache MXNet 和 MPI 运营商。

下载并运行脚本以在集群中安装所需的组件。

```
$ wget -O install_kubeflow.sh https://raw.githubusercontent.com/aws/deep-
learning-containers/master/test/dlc_tests/eks/eks_manifest_templates/kubeflow/
install_kubeflow.sh
$ chmod +x install_kubeflow.sh
$ ./install_kubeflow.sh <EKS_CLUSTER_NAME> <AWS_REGION>
```

Apache MXNet（孵化）分布式 GPU 训练

本教程介绍如何使用参数服务器在多节点 GPU 集群上使用 Apache MXNet（孵化）运行分布式训练。要在 EKS 上运行 MXNet 分布式训练，可以使用Kubernetes mxNet-Operator被命名MXJob. 它将提供一种自定义资源，可让您轻松在 Kubernetes 上运行分布式或非分布式 MXNet 任务（训练和调整）。此操作员已在上一个设置步骤中安装。

使用自定义资源定义 (CRD) 将使用户能够创建和管理 MX 任务，就像 builtin K8s 资源一样。验证是否已安装 MXNet 自定义资源。

```
$ kubectl get crd
```

该输出应包含 `mxjobs.kubeflow.org`。

使用参数服务器运行 MNIST 分布式训练示例

根据可用的集群配置和要运行的作业为您的作业创建 pod 文件 (mx_job_dist.yaml)。您需要指定 3 种 jobModes：调度器、服务器和工作器。您可以指定要使用字段副本生成的 pod 的数量。计划程序、服务器和工作线程的实例类型将是在集群创建时指定的类型。

- 计划程序：只有一个计划程序。计划程序的作用是设置集群。这包括等待每个节点已启动以及节点正在侦听哪个端口的消息。然后，计划程序向所有进程通知集群中的每个其他节点，以便它们可以相互通信。
- 服务器：可以有多个服务器存储模型的参数并与工作人员通信。服务器可能与工作线程进程位于同一位置，也可能位于不同的位置。
- 工作线程实例集 工作节点实际上是在一批训练样本上执行训练。在处理每个批次之前，工作线程从服务器中提取权重。在每个批次后，工作线程还会将梯度发送到服务器。根据模型训练工作负载，在同一计算机上运行多个工作线程可能并不是一个好主意。
- 提供要用于字段映像的容器映像。
- 你可以提供`restartPolicy`来自其中一个 Alwash OnFailure 而且从来没有。它确定 pod 是否在其退出时重新启动。

- 提供要用于字段映像的容器映像。

1. 要创建 MxJob 模板，请根据您的要求修改以下代码块并将其保存在名为的文件中mx_job_dist.yaml.

```
apiVersion: "kubeflow.org/v1beta1"
kind: "MXJob"
metadata:
  name: <JOB_NAME>
spec:
  jobMode: MXTrain
  mxReplicaSpecs:
    Scheduler:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
mxnet-training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
    Server:
      replicas: <NUM_SERVERS>
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
mxnet-training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
    Worker:
      replicas: <NUM_WORKERS>
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
mxnet-training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
              command:
              - "python"
              args:
```

```
            - "/incubator-mxnet/example/image-classification/train_mnist.py"
            - "--num-epochs"
            - <EPOCHS>
            - "--num-layers"
            - <LAYERS>
            - "--kv-store"
            - "dist_device_sync"
            - "--gpus"
            - <GPUS>
            resources:
              limits:
                nvidia.com/gpu: <GPU_LIMIT>
```

2. 使用您刚创建的 pod 文件运行分布式训练任务。

```
$ # Create a job by defining MXJob
kubectl create -f mx_job_dist.yaml
```

3. 列出正在运行的任务。

```
$ kubectl get mxjobs
```

4. 要获取正在运行的任务的状态，请运行以下命令。将 JOB 变量替换为任务的任何名称。

```
$ JOB=<JOB_NAME>
kubectl get mxjobs $JOB -o yaml
```

该输出值应该类似于以下内容：

```
apiVersion: kubeflow.org/v1beta1
kind: MXJob
metadata:
  creationTimestamp: "2020-07-23T16:38:41Z"
  generation: 8
  name: kubeflow-mxnet-gpu-dist-job-3910
  namespace: mxnet-multi-node-training-3910
  resourceVersion: "688398"
  selfLink: /apis/kubeflow.org/v1beta1/namespaces/mxnet-multi-node-training-3910/
mxjobs/kubeflow-mxnet-gpu-dist-job-3910
spec:
  cleanPodPolicy: All
  jobMode: MXTrain
```

```
  mxReplicaSpecs:
    Scheduler:
      replicas: 1
      restartPolicy: Never
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
            name: mxnet
            ports:
            - containerPort: 9091
              name: mxjob-port
            resources: {}
    Server:
      replicas: 2
      restartPolicy: Never
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
            name: mxnet
            ports:
            - containerPort: 9091
              name: mxjob-port
            resources: {}
    Worker:
      replicas: 3
      restartPolicy: Never
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
          - args:
            - /incubator-mxnet/example/image-classification/train_mnist.py
            - --num-epochs
            - "20"
            - --num-layers
```

```
            - "2"
            - --kv-store
            - dist_device_sync
            - --gpus
            - "0"
            command:
            - python
            image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
            name: mxnet
            ports:
            - containerPort: 9091
              name: mxjob-port
            resources:
              limits:
                nvidia.com/gpu: "1"
status:
  conditions:
  - lastTransitionTime: "2020-07-23T16:38:41Z"
    lastUpdateTime: "2020-07-23T16:38:41Z"
    message: MXJob kubeflow-mxnet-gpu-dist-job-3910 is created.
    reason: MXJobCreated
    status: "True"
    type: Created
  - lastTransitionTime: "2020-07-23T16:38:41Z"
    lastUpdateTime: "2020-07-23T16:40:50Z"
    message: MXJob kubeflow-mxnet-gpu-dist-job-3910 is running.
    reason: MXJobRunning
    status: "True"
    type: Running
  mxReplicaStatuses:
    Scheduler:
      active: 1
    Server:
      active: 2
    Worker:
      active: 3
  startTime: "2020-07-23T16:40:50Z"
```

Note

状态提供有关资源的状态的信息。
阶段-表示作业的阶段，将是 “创建”、“运行”、 CleanUp、失败或完成。

状态-提供作业的总体状态，将为 “正在运行”、“成功” 或 “失败”。

5. 如果要删除任务，请将目录更改为您启动任务的位置并运行以下命令：

```
$ kubectl delete -f mx_job_dist.yaml
```

使用 Horovod 分布式 GPU 训练的 Apache MXNet（孵化中）

本教程介绍如何在您的多节点 GPU 集群上设置 Apache MXNet（孵化）模型的分布式训练霍罗沃德. 它使用已包含训练脚本的示例映像，并且将一个 3 节点集群与 node-type=p3.8xlarge 结合使用。本教程运行Horovod 示例脚本适用于 MNIST 模型上的 MXNet。

1. 验证 miJob 自定义资源是否已安装。

```
$ kubectl get crd
```

该输出应包含 mpijobs.kubeflow.org。

2. 创建 MPI Job 模板并定义节点数量（副本）和每个节点拥有的 GPU 数量（gpusPerReplica)。根据您的要求修改以下代码块并将其保存在名为的文件中mx-mnist-horovod-job.yaml.

```
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: <JOB_NAME>
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
            name: <JOB_NAME>
            args:
            - --epochs
            - "10"
            - --lr
```

```
            - "0.001"
            command:
            - mpirun
            - -mca
            - btl_tcp_if_exclude
            - lo
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - --bind-to
            - none
            - -map-by
            - slot
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -x
            - NCCL_SOCKET_IFNAME=eth0
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - MXNET_CUDNN_AUTOTUNE_DEFAULT=0
            - python
            - /horovod/examples/mxnet_mnist.py
    Worker:
      replicas: <NUM_WORKERS>
      template:
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.8.0-gpu-py37-cu110-ubuntu16.04-example
            name: mpi-worker
            resources:
              limits:
                nvidia.com/gpu: <GPUS>
```

3. 使用您刚创建的 pod 文件运行分布式训练任务。

```
$ kubectl create -f mx-mnist-horovod-job.yaml
```

4. 检查状态。任务名称显示在状态中。如果你正在运行任何其他测试或者之前运行过某项测试，它会出现在这个列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods -o wide
```

您应该可以看到类似于如下输出的内容：

```
NAME                                     READY   STATUS    RESTARTS AGE

mxnet-mnist-horovod-job-716-launcher-4wc7f   1/1    Running    0       31s
mxnet-mnist-horovod-job-716-worker-0         1/1    Running    0       31s
mxnet-mnist-horovod-job-716-worker-1         1/1    Running    0       31s
mxnet-mnist-horovod-job-716-worker-2         1/1    Running    0       31s
```

5. 根据上述启动程序 pod 的名称，检查日志以查看训练输出。

```
$ kubectl logs -f --tail 10 <LAUNCHER_POD_NAME>
```

6. 您可以检查日志以观察训练进度。您还可以继续检查“get pods”以刷新状态。当状态变为“Completed (已完成)”时，您将知道该训练任务已完成。
7. 要清理并重新运行作业，请执行以下操作：

```
$ kubectl delete -f mx-mnist-horovod-job.yaml
```

后续步骤

要在 Amazon EKS 上使用带有Deep Learning Containers 的 MXNet 学习基于 GPU 的推理，请参阅Apache MxNet（孵化）GPU 推理.

TensorFlow使用 Horovod 分布式 GPU 训练

本教程介绍如何设置分布式训练 TensorFlow 您的多节点 GPU 集群上使用的模型霍罗沃德. 它使用已包含训练脚本的示例映像，并且将一个 3 节点集群与 `node-type=p3.16xlarge` 结合使用。您可以将本教程与以下任一方法结合使用 TensorFlow 要么 TensorFlow 2. 要与之配合使用 TensorFlow 2，将 Docker 镜像更改为 TensorFlow 2 张图片。

1. 验证 miJob 自定义资源是否已安装。

```
$ kubectl get crd
```

该输出应包含 mpijobs.kubeflow.org。

2. 创建 MPI Job 模板并定义节点数量（副本）和每个节点拥有的 GPU 数量（gpusPerReplica)。根据您的要求修改以下代码块并将其保存在名为的文件中tf-resnet50-horovod-job.yaml.

```
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: <JOB_NAME>
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         spec:
           containers:
           - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
tensorflow-training:1.15.5-gpu-py37-cu100-ubuntu18.04-example
             name: <JOB_NAME>
             command:
              - mpirun
              - -mca
              - btl_tcp_if_exclude
              - lo
              - -mca
              - pml
              - ob1
              - -mca
              - btl
              - ^openib
              - --bind-to
              - none
              - -map-by
              - slot
              - -x
              - LD_LIBRARY_PATH
              - -x
              - PATH
              - -x
              - NCCL_SOCKET_IFNAME=eth0
              - -x
```

```
              - NCCL_DEBUG=INFO
              - -x
              - MXNET_CUDNN_AUTOTUNE_DEFAULT=0
              - python
              - /deep-learning-models/models/resnet/tensorflow/
train_imagenet_resnet_hvd.py
             args:
              - --num_epochs
              - "10"
              - --synthetic
    Worker:
      replicas: <NUM_WORKERS>
      template:
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
tensorflow-training:1.15.5-gpu-py37-cu100-ubuntu18.04-example
            name: tensorflow-worker
            resources:
              limits:
                nvidia.com/gpu: <GPUS>
```

3. 使用您刚创建的 pod 文件运行分布式训练作业。

```
$ kubectl create -f tf-resnet50-horovod-job.yaml
```

4. 检查状态。任务名称显示在状态中。如果您正在运行任何其他测试或之前运行过其他测试，则它们会出现在此列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods -o wide
```

您应该可以看到类似于如下输出的内容：

```
NAME                                          READY   STATUS    RESTARTS AGE

tf-resnet50-horovod-job-1794-launcher-9wbsg   1/1     Running     0       31s
tf-resnet50-horovod-job-1794-worker-0         1/1     Running     0       31s
tf-resnet50-horovod-job-1794-worker-1         1/1     Running     0       31s
tf-resnet50-horovod-job-1794-worker-2         1/1     Running     0       31s
```

5. 根据上述启动程序 pod 的名称，检查日志以查看训练输出。

```
$ kubectl logs -f --tail 10 <LAUNCHER_POD_NAME>
```

6. 您可以检查日志以观察训练进度。您还可以继续检查“get pods”以刷新状态。当状态变为“Completed (已完成)”时，您将知道该训练任务已完成。
7. 要清理并重新运行作业，请执行以下操作：

```
$ kubectl delete -f tf-resnet50-horovod-job.yaml
```

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，请使用 TensorFlow 有关DDeep Learning Containers ContaTensorFlow GPU 推理.

PyTorch分布式 GU 训练

本教程将指导您进行分布式训练 PyTorch 在您的多节点 GPU 集群上。它使用 Gloo 作为后端。

1. 验证 PyTorch 已安装自定义资源。

```
$ kubectl get crd
```

该输出应包含 pytorchjobs.kubeflow.org。

2. 确保 NVIDIA 插件daemonset正在运行。

```
$ kubectl get daemonset -n kubeflow
```

输出值应该与下面类似。

```
NAME                           DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE
 SELECTOR AGE
nvidia-device-plugin-daemonset   3         3      3        3        3    <none>
 35h
```

3. 使用以下文本创建基于 Gloo 的分布式数据并行作业。将其保存在名为的文件中distributed.yaml.

```
apiVersion: kubeflow.org/v1
```

```
kind: PyTorchJob
metadata:
  name: "kubeflow-pytorch-gpu-dist-job"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
pytorch-training:1.7.1-gpu-py36-cu110-ubuntu18.04-example"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
    Worker:
      replicas: 2
      restartPolicy: OnFailure
      template:
        spec:
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
pytorch-training:1.7.1-gpu-py36-cu110-ubuntu18.04-example"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
            resources:
              limits:
                nvidia.com/gpu: 1
```

4. 使用您刚创建的 pod 文件运行分布式训练任务。

```
$ kubectl create -f distributed.yaml
```

5. 您可以使用以下方法检查作业的状态：

```
$ kubectl logs kubeflow-pytorch-gpu-dist-job
```

要连续查看日志，请使用：

```
$ kubectl logs -f <pod>
```

请参阅清除了解有关在使用完集群后清理集群的信息。

Amazon S3 插件适用于 PyTorch

Deep Learning Containers 包含一个插件，该插件使您可以将来自 Amazon S3 桶的数据用于 PyTorch 训练。查看Amazon EKSAmazon S3 插件适用于 PyTorch GU 指南开始使用。

有关更多信息和其他示例，请参阅Amazon S3 插件适用于 PyTorch存储库。

请参阅清除了解有关在使用完集群后清理集群的信息。

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，请使用 PyTorch 有关DDeep Learning Containers ContaPyTorch GPU 推理.

Inference

使用中的步骤创建群集后Amazon EK 设置，你可以用它来运行推理作业。为了推断，您可以使用 CPU 或 GPU 示例，具体取决于集群中的节点。推理仅支持单节点配置。以下主题介绍了如何运行推理AWS 使用 Apache MXNet（孵化）、PyTorch、TensorFlow 和 TensorFlow 2 在 EKS 上的 Deep Learning Containers。

目录

CPU 推理

本部分指导您使用 Apache MXNet（孵化）、PyTorch、TensorFlow 和 TensorFlow 2 在 EKS CPU 集群的 Deep Learning Containers 上运行推理。

有关 Deep Learning Containers 的完整列表，请参阅。可用的 Deep Learning Containers 映像.

> **Note**
>
> 如果您是在使用 MKL，请参阅。AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

目录

Apache MxNet（孵化）CPU 推理

在本教程中，创建一个 Kubernetes 服务和部署，用于使用 MXNet 来运行 CPU 推理。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类型`ServiceTypes`. 默认`ServiceType`是`ClusterIP`. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 创建命名空间。您可能需要更改 kubeconfig 以指向正确的集群。验证您已设置 “training-cpu-1” 或将其更改为您的 CPU 集群的配置。有关如何设置群集的更多信息，请参阅。Amazon EK 设置.

   ```
   $ NAMESPACE=mx-inference; kubectl –kubeconfig=/home/ubuntu/.kube/eksctl/clusters/
   training-cpu-1 create namespace ${NAMESPACE}
   ```

2. （使用公共模型时的可选步骤。）在可装载的网络位置（如 Amazon S3）处设置您的模型。有关如何将训练后的模型上传到 S3，请参阅。TensorFlow CPU 推理. 将密钥应用于您的命名空间。有关密钥的更多信息，请参阅Kubernetes 密钥文档.

   ```
   $ kubectl -n ${NAMESPACE} apply -f secret.yaml
   ```

3. 使用以下内容创建名为 `mx_inference.yaml` 的文件。此示例文件指定了模型、使用的 MxNet 推理图像以及模型的位置。此示例使用公共模型，因此您无需修改它。

   ```
   ---
   kind: Service
   apiVersion: v1
   metadata:
   ```

```
  name: squeezenet-service
  labels:
    app: squeezenet-service
spec:
  ports:
  - port: 8080
    targetPort: mms
  selector:
    app: squeezenet-service
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: squeezenet-service
  labels:
    app: squeezenet-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: squeezenet-service
  template:
    metadata:
      labels:
        app: squeezenet-service
    spec:
      containers:
      - name: squeezenet-service
        image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.6.0-
cpu-py36-ubuntu16.04
        args:
        - mxnet-model-server
        - --start
        - --mms-config /home/model-server/config.properties
        - --models squeezenet=https://s3.amazonaws.com/model-server/
model_archive_1.0/squeezenet_v1.1.mar
        ports:
        - name: mms
          containerPort: 8080
        - name: mms-management
          containerPort: 8081
        imagePullPolicy: IfNotPresent
```

4. 将配置应用于之前定义的命名空间中的新 pod。

```
$ kubectl -n ${NAMESPACE} apply -f mx_inference.yaml
```

您的输出应类似于以下内容：

```
service/squeezenet-service created
deployment.apps/squeezenet-service created
```

5. 检查 pod 的状态。

```
$ kubectl get pods -n ${NAMESPACE}
```

重复状态检查，直到您看到以下 “RUNNING” 状态：

```
NAME                        READY     STATUS    RESTARTS   AGE
squeezenet-service-xvw1  1/1       Running   0          3m
```

6. 要进一步描述 pod，请运行以下命令：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

7. 由于此处的 serviceType 为 ClusterIP，您可以使用以下命令将端口从您的容器转发至您的主机：

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=squeezenet-service -o jsonpath='{.items[0].metadata.name}'` 8080:8080
 &
```

8. 下载小猫的图像。

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
```

9. 使用小猫的图像对模型运行推理过程：

```
$ curl -X POST http://127.0.0.1:8080/predictions/squeezenet -T kitten.jpg
```

TensorFlow CPU 推理

在本教程中，创建一个 Kubernetes 服务和部署，用于使用 TensorFlow 来运行 CPU 推理。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类

型ServiceTypes. 默认ServiceType是ClusterIP. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 创建命名空间。您可能需要更改 kubeconfig 以指向正确的集群。验证您已设置 “training-cpu-1” 或将其更改为您的 CPU 集群的配置。有关如何设置群集的更多信息，请参阅。Amazon EK 设置.

```
$ NAMESPACE=tf-inference; kubectl –kubeconfig=/home/ubuntu/.kube/eksctl/clusters/
training-cpu-1 create namespace ${NAMESPACE}
```

2. 用于推理的模型可通过不同的方式进行检索，例如，使用共享卷和 Amazon S3。由于 Kubernetes 服务需要访问 Amazon S3 和 Amazon ECR，因此您必须存储AWS凭据作为 Kubernetes 秘密。在本示例中，请使用 S3 来存储和获取训练后的模型。

   验证您的AWS凭证。他们必须具有 S3 写入访问权限。

```
$ cat ~/.aws/credentials
```

3. 该输出值将类似于以下内容：

```
$ [default]
aws_access_key_id = YOURACCESSKEYID
aws_secret_access_key = YOURSECRETACCESSKEY
```

4. 使用 base64 对这些凭证进行编码。

   首先编码访问密钥。

```
$ echo -n 'YOURACCESSKEYID' | base64
```

   接下来编码秘密访问密钥。

```
$ echo -n 'YOURSECRETACCESSKEY' | base64
```

   您的输出应类似于以下内容：

```
$ echo -n 'YOURACCESSKEYID' | base64
RkFLRUFXU0FDQ0VTU0tFWUlE
$ echo -n 'YOURSECRETACCESSKEY' | base64
RkFLRUFXU1NFQ1JFVEFDQ0VTU0tFWQ==
```

5. 创建一个名为的文件secret.yaml并将以下内容置于您的主目录中。该文件用于存储密钥。

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-s3-secret
type: Opaque
data:
  AWS_ACCESS_KEY_ID: YOURACCESSKEYID
  AWS_SECRET_ACCESS_KEY: YOURSECRETACCESSKEY
```

6. 将密钥应用于您的命名空间。

```
$ kubectl -n ${NAMESPACE} apply -f secret.yaml
```

7. 克隆tensorflow 服务存储库。

```
$ git clone https://github.com/tensorflow/serving/
$ cd serving/tensorflow_serving/servables/tensorflow/testdata/
```

8. 同步预训练后的saved_model_half_plus_two_cpu将模型到您的 S3 存储桶。

```
$ aws s3 sync saved_model_half_plus_two_cpu s3://<your_s3_bucket>/
saved_model_half_plus_two
```

9. 使用以下内容创建名为 tf_inference.yaml 的文件。更新--model_base_path以使用您的 S3 存储桶。您可以将它与 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow2 一起使用，请将 Docker 映像更改为 TensorFlow 2 映像。

```
---
  kind: Service
  apiVersion: v1
  metadata:
    name: half-plus-two
    labels:
      app: half-plus-two
  spec:
    ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
```

```
      port: 9000
      targetPort: 9000
    selector:
      app: half-plus-two
      role: master
    type: ClusterIP
  ---
  kind: Deployment
  apiVersion: apps/v1
  metadata:
    name: half-plus-two
    labels:
      app: half-plus-two
      role: master
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: half-plus-two
        role: master
    template:
      metadata:
        labels:
          app: half-plus-two
          role: master
      spec:
        containers:
        - name: half-plus-two
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-cpu-py36-ubuntu18.04
          command:
          - /usr/bin/tensorflow_model_server
          args:
          - --port=9000
          - --rest_api_port=8500
          - --model_name=saved_model_half_plus_two
          - --model_base_path=s3://tensorflow-trained-models/
saved_model_half_plus_two
          ports:
          - containerPort: 8500
          - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
          - name: AWS_ACCESS_KEY_ID
```

```
            valueFrom:
              secretKeyRef:
                key: AWS_ACCESS_KEY_ID
                name: aws-s3-secret
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                key: AWS_SECRET_ACCESS_KEY
                name: aws-s3-secret
          - name: AWS_REGION
            value: us-east-1
          - name: S3_USE_HTTPS
            value: "true"
          - name: S3_VERIFY_SSL
            value: "true"
          - name: S3_ENDPOINT
            value: s3.us-east-1.amazonaws.com
```

10. 将配置应用于之前定义的命名空间中的新 pod。

```
$ kubectl -n ${NAMESPACE} apply -f tf_inference.yaml
```

您的输出应类似于以下内容：

```
service/half-plus-two created
deployment.apps/half-plus-two created
```

11. 检查 pod 的状态。

```
$ kubectl get pods -n ${NAMESPACE}
```

重复状态检查，直到您看到以下 “RUNNING” 状态：

```
NAME                        READY     STATUS    RESTARTS   AGE
half-plus-two-vmwp9  1/1       Running   0          3m
```

12. 要进一步描述 pod，您可以运行：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

13. 由于 serviceType 为 ClusterIP，您可以将端口从您的容器转发至您的主机。

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=half-plus-two -o jsonpath='{.items[0].metadata.name}'` 8500:8500 &
```

14. 将以下 json 字符串置于名为的文件中half_plus_two_input.json

```
{"instances": [1.0, 2.0, 5.0]}
```

15. 在模型上运行推理模型。

```
$ curl -d @half_plus_two_input.json -X POST http://localhost:8500/v1/models/
saved_model_half_plus_two_cpu:predict
```

您的输出应与以下内容类似：

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

PyTorch CPU 推理

在此方法中，创建一个 Kubernetes 服务和部署，用于使用 PyTorch 运行 CPU 推理。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类型ServiceTypes. 默认ServiceType是ClusterIP. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 创建命名空间。您可能需要更改 kubeconfig 以指向正确的集群。验证您已设置 “training-cpu-1” 或将其更改为您的 CPU 集群的配置。有关如何设置群集的更多信息，请参阅。Amazon EK 设置.

```
$ NAMESPACE=pt-inference; kubectl create namespace ${NAMESPACE}
```

2. （使用公共模型时的可选步骤。）在可装载的网络位置（如 Amazon S3）处设置您的模型。有关如何将训练后的模型上传到 S3，请参阅。TensorFlow CPU 推理. 将密钥应用于您的命名空间。有关密钥的更多信息，请参阅Kubernetes 密钥文档.

```
$ kubectl -n ${NAMESPACE} apply -f secret.yaml
```

3. 使用以下内容创建名为 pt_inference.yaml 的文件。此示例文件指定了模型、所使用的 PyTorch 推理图像以及模型的位置。此示例使用公共模型，因此您无需修改它。

```
---
kind: Service
apiVersion: v1
metadata:
  name: densenet-service
  labels:
    app: densenet-service
spec:
  ports:
  - port: 8080
    targetPort: mms
  selector:
    app: densenet-service
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: densenet-service
  labels:
    app: densenet-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: densenet-service
  template:
    metadata:
      labels:
        app: densenet-service
    spec:
      containers:
      - name: densenet-service
        image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
inference:1.3.1-cpu-py36-ubuntu16.04
        args:
        - mxnet-model-server
        - --start
        - --mms-config /home/model-server/config.properties
        - --models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/multi-
model-server/densenet/densenet.mar
        ports:
        - name: mms
          containerPort: 8080
```

```
        - name: mms-management
          containerPort: 8081
        imagePullPolicy: IfNotPresent
```

4. 将配置应用于之前定义的命名空间中的新 pod。

```
$ kubectl -n ${NAMESPACE} apply -f pt_inference.yaml
```

您的输出应类似于以下内容：

```
service/densenet-service created
deployment.apps/densenet-service created
```

5. 检查 pod 的状态并等待 pod 处于 “RUNNING (正在运行)” 状态：

```
$ kubectl get pods -n ${NAMESPACE} -w
```

您的输出应类似于以下内容：

```
NAME                    READY     STATUS    RESTARTS   AGE
densenet-service-xvw1   1/1       Running   0          3m
```

6. 要进一步描述 pod，请运行以下命令：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

7. 由于此处的 serviceType 为 ClusterIP，您可以将端口从您的容器转发至您的主机。

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=densenet-service -o jsonpath='{.items[0].metadata.name}'` 8080:8080 &
```

8. 在启动您的服务器后，现在您可以使用以下命令从不同的窗口来运行推理：

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://127.0.0.1:8080/predictions/densenet -T flower.jpg
```

使用完集群后，请参阅 EKS 清除以获取有关清除集群的信息。

后续步骤

要了解如何在 Amazon EKS 上将自定义入口点与 Deep Learning Containers 结合使用，请参阅自定义入口点.

GPU 推理

本部分介绍如何使用 Apache MXNet（孵化）、PyTorch、TensorFlow 和 TensorFlow 2 在 EKS GPU 集群的 Deep Learning Containers 上运行推理。

有关 Deep Learning Containers 的完整列表，请参阅。可用的 Deep Learning Containers 映像.

> **Note**
>
> MKL 用户：读取 AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议以获得最佳训练或推理性能。

目录

Apache MxNet（孵化）GPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类型`ServiceTypes`. 默认`ServiceType`是`ClusterIP`. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 对于基于 GPU 的推理，安装适用于 Kubernetes 的 NVIDIA 设备插件：

   ```
   $ kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/
   v1.12/nvidia-device-plugin.yml
   ```

2. 验证 nvidia-device-plugin-daemonset 是否正在正确运行。

   ```
   $ kubectl get daemonset -n kube-system
   ```

   该输出值将类似于以下内容：

```
NAME                              DESIRED   CURRENT   READY     UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-node                          3         3         3         3           3
     <none>          6d
kube-proxy                        3         3         3         3           3
     <none>          6d
nvidia-device-plugin-daemonset    3         3         3         3           3
     <none>          57s
```

3. 创建命名空间。您可能需要更改 kubeconfig 以指向正确的集群。验证您已设置“training-gpu-1”或将其更改为您的 GPU 集群的配置。有关如何设置群集的更多信息，请参阅。[Amazon EK 设置](#).

```
$ NAMESPACE=mx-inference; kubectl –kubeconfig=/home/ubuntu/.kube/eksctl/clusters/
training-gpu-1 create namespace ${NAMESPACE}
```

4. （使用公共模型时的可选步骤。）在可装载的网络位置（如 S3）处设置您的模型。请参阅这些步骤以将训练后的模型上传到“利用 TensorFlow 进行推理”部分中提及的 S3。将密钥应用于您的命名空间。有关密钥的更多信息，请参阅[Kubernetes 密钥文档](#).

```
$ kubectl -n ${NAMESPACE} apply -f secret.yaml
```

5. 创建 `mx_inference.yaml` 文件。使用下一个代码块的内容作为其内容。

```
---
kind: Service
apiVersion: v1
metadata:
  name: squeezenet-service
  labels:
    app: squeezenet-service
spec:
  ports:
  - port: 8080
    targetPort: mms
  selector:
    app: squeezenet-service
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: squeezenet-service
```

```
  labels:
    app: squeezenet-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: squeezenet-service
  template:
    metadata:
      labels:
        app: squeezenet-service
    spec:
      containers:
      - name: squeezenet-service
        image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.6.0-
gpu-py36-cu101-ubuntu16.04
        args:
        - mxnet-model-server
        - --start
        - --mms-config /home/model-server/config.properties
        - --models squeezenet=https://s3.amazonaws.com/model-server/
model_archive_1.0/squeezenet_v1.1.mar
        ports:
        - name: mms
          containerPort: 8080
        - name: mms-management
          containerPort: 8081
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: 4
            memory: 4Gi
            nvidia.com/gpu: 1
          requests:
            cpu: "1"
            memory: 1Gi
```

6. 将配置应用于之前定义的命名空间中的新 pod：

```
$ kubectl -n ${NAMESPACE} apply -f mx_inference.yaml
```

您的输出应类似于以下内容：

```
service/squeezenet-service created
deployment.apps/squeezenet-service created
```

7. 检查 pod 的状态并等待 pod 处于“RUNNING (正在运行)”状态：

```
$ kubectl get pods -n ${NAMESPACE}
```

8. 重复检查状态步骤，直到您看到以下“RUNNING (正在运行)”状态：

```
NAME                        READY     STATUS    RESTARTS   AGE
squeezenet-service-xvw1  1/1       Running   0          3m
```

9. 要进一步描述 pod，您可以运行：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

10. 由于此处的 serviceType 为 ClusterIP，您可以将端口从您的容器转发至您的主机，（& 将在后台中运行此项）：

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=squeezenet-service -o jsonpath='{.items[0].metadata.name}'` 8080:8080
 &
```

11. 下载小猫的图像：

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
```

12. 在该模型上运行推理：

```
$ curl -X POST http://127.0.0.1:8080/predictions/squeezenet -T kitten.jpg
```

TensorFlow GPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类型`ServiceTypes`. 默认`ServiceType`是`ClusterIP`. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 对于基于 GPU 的推理，安装适用于 Kubernetes 的 NVIDIA 设备插件：

```
$ kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/
v1.12/nvidia-device-plugin.yml
```

2. 验证 nvidia-device-plugin-daemonset 是否正在正确运行。

```
$ kubectl get daemonset -n kube-system
```

该输出值将类似于以下内容：

```
NAME                                 DESIRED   CURRENT   READY     UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-node                             3         3         3         3            3
     <none>          6d
kube-proxy                           3         3         3         3            3
     <none>          6d
nvidia-device-plugin-daemonset   3         3         3         3            3
     <none>          57s
```

3. 创建命名空间。您可能需要更改 kubeconfig 以指向正确的集群。验证您已设置“training-gpu-1”或将其更改为您的 GPU 集群的配置。有关如何设置群集的更多信息，请参阅。Amazon EK 设置.

```
$ NAMESPACE=tf-inference; kubectl –kubeconfig=/home/ubuntu/.kube/eksctl/clusters/
training-gpu-1 create namespace ${NAMESPACE}
```

4. 用于推理的模型可通过不同的方式进行检索，例如，使用共享卷、S3 等。由于该服务需要访问 S3 和 ECR，您必须存储您的AWS凭据作为 Kubernetes 秘密。在本示例中，您将使用 S3 来存储和提取训练后的模型。

检查您的AWS凭证。这些凭证必须具有 S3 写入访问权限。

```
$ cat ~/.aws/credentials
```

5. 输出将类似于以下内容：

```
$ [default]
aws_access_key_id = FAKEAWSACCESSKEYID
aws_secret_access_key = FAKEAWSSECRETACCESSKEY
```

6. 使用 base64 对这些凭证进行编码。首先编码访问密钥。

```
$ echo -n 'FAKEAWSACCESSKEYID' | base64
```

接下来编码秘密访问密钥。

```
$ echo -n 'FAKEAWSSECRETACCESSKEYID' | base64
```

您的输出应类似于以下内容：

```
$ echo -n 'FAKEAWSACCESSKEYID' | base64
RkFLRUFXU0FDQ0VTU0tFWUlE
$ echo -n 'FAKEAWSSECRETACCESSKEY' | base64
RkFLRUFXU1NFQ1JFVEFDQ0VTU0tFWQ==
```

7. 创建 yaml 文件来存储密钥。在您的主目录中将该密钥另存为 secret.yaml。

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-s3-secret
type: Opaque
data:
  AWS_ACCESS_KEY_ID: RkFLRUFXU0FDQ0VTU0tFWUlE
  AWS_SECRET_ACCESS_KEY: RkFLRUFXU1NFQ1JFVEFDQ0VTU0tFWQ==
```

8. 将密钥应用于您的命名空间：

```
$ kubectl -n ${NAMESPACE} apply -f secret.yaml
```

9. 在本示例中，您将克隆 tensorflow-serving 存储库并将预训练模型同步到 S3 存储桶。以下示例命名存储桶 `tensorflow-serving-models`。它还将已保存的模型同步到名 `saved_model_half_plus_two_gpu` 的 S3 存储桶。

```
$ git clone https://github.com/tensorflow/serving/
$ cd serving/tensorflow_serving/servables/tensorflow/testdata/
```

10. 同步 CPU 模型。

```
$ aws s3 sync saved_model_half_plus_two_gpu s3://<your_s3_bucket>/
saved_model_half_plus_two_gpu
```

11. 创建 tf_inference.yaml 文件。使用下一个代码块的内容作为其内容，并将 --model_base_path 更新为使用您的 S3 存储桶。您可以将它与 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow2 一起使用，请将 Docker 映像更改为 TensorFlow 2 映像。

```
---
  kind: Service
  apiVersion: v1
  metadata:
    name: half-plus-two
    labels:
      app: half-plus-two
  spec:
    ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
    selector:
      app: half-plus-two
      role: master
    type: ClusterIP
---
  kind: Deployment
  apiVersion: apps/v1
  metadata:
    name: half-plus-two
    labels:
      app: half-plus-two
      role: master
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: half-plus-two
        role: master
    template:
      metadata:
        labels:
          app: half-plus-two
          role: master
      spec:
```

```
        containers:
        - name: half-plus-two
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-gpu-py36-cu100-ubuntu18.04
          command:
          - /usr/bin/tensorflow_model_server
          args:
          - --port=9000
          - --rest_api_port=8500
          - --model_name=saved_model_half_plus_two_gpu
          - --model_base_path=s3://tensorflow-trained-models/
saved_model_half_plus_two_gpu
          ports:
          - containerPort: 8500
          - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                key: AWS_ACCESS_KEY_ID
                name: aws-s3-secret
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                key: AWS_SECRET_ACCESS_KEY
                name: aws-s3-secret
          - name: AWS_REGION
            value: us-east-1
          - name: S3_USE_HTTPS
            value: "true"
          - name: S3_VERIFY_SSL
            value: "true"
          - name: S3_ENDPOINT
            value: s3.us-east-1.amazonaws.com
            resources:
              limits:
                cpu: 4
                memory: 4Gi
                nvidia.com/gpu: 1
              requests:
                cpu: "1"
                memory: 1Gi
```

12. 将配置应用于之前定义的命名空间中的新 pod：

```
$ kubectl -n ${NAMESPACE} apply -f tf_inference.yaml
```

您的输出应类似于以下内容：

```
service/half-plus-two created
deployment.apps/half-plus-two created
```

13. 检查 pod 的状态并等待 pod 处于“RUNNING (正在运行)”状态：

```
$ kubectl get pods -n ${NAMESPACE}
```

14. 重复检查状态步骤，直到您看到以下“RUNNING (正在运行)”状态：

```
NAME                        READY     STATUS    RESTARTS   AGE
half-plus-two-vmwp9  1/1       Running   0          3m
```

15. 要进一步描述 pod，您可以运行：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

16. 由于此处的 serviceType 为 ClusterIP，您可以将端口从您的容器转发至您的主机，（& 将在后台中运行此项）：

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=half-plus-two -o jsonpath='{.items[0].metadata.name}'` 8500:8500 &
```

17. 将以下 json 字符串置于名为 `half_plus_two_input.json` 的文件中

```
{"instances": [1.0, 2.0, 5.0]}
```

18. 在该模型上运行推理：

```
$ curl -d @half_plus_two_input.json -X POST http://localhost:8500/v1/models/
saved_model_half_plus_two_cpu:predict
```

预期的输出如下所示：

```
{
    "predictions": [2.5, 3.0, 4.5
    ]
}
```

PyTorch GPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。Kubernetes 服务公开了一个进程及其端口。创建 Kubernetes 服务时，可以指定要使用的服务类型ServiceTypes. 默认ServiceType是ClusterIP. 部署负责确保一定数量的 Pod 始终启动并运行。

1. 对于基于 GPU 的推理，安装适用于 Kubernetes 的 NVIDIA 设备插件。

```
$ kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/
v1.12/nvidia-device-plugin.yml
```

2. 验证 nvidia-device-plugin-daemonset 是否正在正确运行。

```
$ kubectl get daemonset -n kube-system
```

该输出值将类似于以下内容。

```
NAME                             DESIRED   CURRENT   READY     UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-node                         3         3         3         3            3
     <none>          6d
kube-proxy                       3         3         3         3            3
     <none>          6d
nvidia-device-plugin-daemonset   3         3         3         3            3
     <none>          57s
```

3. 创建命名空间。

```
$ NAMESPACE=pt-inference; kubectl create namespace ${NAMESPACE}
```

4. （使用公共模型时的可选步骤。）在可装载的网络位置（如 S3）处设置您的模型。请参阅这些步骤以将训练后的模型上传到“利用 TensorFlow 进行推理”部分中提及的 S3。将密钥应用于您的命名空间。有关密钥的更多信息，请参阅Kubernetes 密钥文档.

```
$ kubectl -n ${NAMESPACE} apply -f secret.yaml
```

5. 创建 pt_inference.yaml 文件。使用下一个代码块的内容作为其内容。

```
---
kind: Service
apiVersion: v1
metadata:
  name: densenet-service
  labels:
    app: densenet-service
spec:
  ports:
  - port: 8080
    targetPort: mms
  selector:
    app: densenet-service
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: densenet-service
  labels:
    app: densenet-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: densenet-service
  template:
    metadata:
      labels:
        app: densenet-service
    spec:
      containers:
      - name: densenet-service
        image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
inference:1.3.1-gpu-py36-cu101-ubuntu16.04"
        args:
        - mxnet-model-server
        - --start
        - --mms-config /home/model-server/config.properties
```

```
        - --models densenet=https://dlc-samples.s3.amazonaws.com/pytorch/multi-
model-server/densenet/densenet.mar
        ports:
        - name: mms
          containerPort: 8080
        - name: mms-management
          containerPort: 8081
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: 4
            memory: 4Gi
            nvidia.com/gpu: 1
          requests:
            cpu: "1"
            memory: 1Gi
```

6. 将配置应用于之前定义的命名空间中的新 pod。

```
$ kubectl -n ${NAMESPACE} apply -f pt_inference.yaml
```

您的输出应类似于以下内容：

```
service/densenet-service created
deployment.apps/densenet-service created
```

7. 检查 pod 的状态并等待 pod 处于“RUNNING (正在运行)”状态。

```
$ kubectl get pods -n ${NAMESPACE}
```

您的输出应类似于以下内容：

```
NAME                    READY     STATUS    RESTARTS   AGE
densenet-service-xvw1   1/1       Running   0          3m
```

8. 要进一步描述 pod，您可以运行：

```
$ kubectl describe pod <pod_name> -n ${NAMESPACE}
```

9. 由于此处的 serviceType 为 ClusterIP，您可以将端口从您的容器转发至您的主机（& 将在后台中运行此项）。

```
$ kubectl port-forward -n ${NAMESPACE} `kubectl get pods -n ${NAMESPACE} --
selector=app=densenet-service -o jsonpath='{.items[0].metadata.name}'` 8080:8080 &
```

10. 在启动您的服务器后，现在您可以从不同的窗口来运行推理。

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/flower.jpg
curl -X POST http://127.0.0.1:8080/predictions/densenet -T flower.jpg
```

使用完集群后，请参阅 EKS 清除以获取有关清除集群的信息。

后续步骤

要了解如何在 Amazon EKS 上将自定义入口点与 Deep Learning Containers 结合使用，请参阅自定义入口点.

AWS安装时的 Kubeflow

本节提供安装说明，以使用开启 Kubeflow 的Deep Learning Containers（Kubeflow 的开源发行版）设置AWS深度学习环境。AWS完成AWS安装 Kubeflow 后，您可以继续阅读本系列的培训教程。

在 Kubeflow 上部署AWS

要在上部署 KubeflowAWS，请按照 KubeflowAWS 文档中的 Vanilla 部署选项进行操作。确保满足所有先决条件。安装说明指导您在上部署 Kubeflow 之前创建 Amazon EKS 集群AWS。

如果您按照前面的说明部署了 GPU 集群，则已安装适用于 Kubernetes 的 NVIDIA 设备插件。您不需要任何额外设置。

> **Note**
>
> 以下教程使用原版的 Kubeflow onAWS 作为示例。但是，您可以使用 Kubeflow 的任何其他部署选项来运行此 Kubeflow onAWS 部分中的所有训练和推理教程AWS。
> 有关在AWS部署 Kubeflow 时设置和配置 Amazon RDS、Amazon S3 和 Amazon Cognito 资源的信息，请参阅 KubeflowAWS 文档中的部署选项。

设置完您的 Amazon EKS 集群后，您可以在下一节中验证您的上下文是否指向您的集群。

验证集群连接

以下步骤显示如何验证您的环境。这是为了确保您与正确的集群进行交互。

1. 首先，通过运行以下命令确认集群处于活动状态。

```
aws eks --region <region> describe-cluster --name <cluster-name> --query
 cluster.status
```

您应当看到如下输出。

```
"ACTIVE"
```

2. 要检查您的当前上下文，请运行此命令。输出中的current-context字段应包含您的集群名称。

```
kubectl config view
```

如果您current-context不是要与之交互的集群，请运行以下命令对其进行更新。有关更新您的的更多信息kubeconfig，请访问 Amazon EKS 文档

```
aws eks update-kubeconfig --region <region> --name <cluster-name>
```

在您部署 KubeflowAWS 并更新了当前上下文后，请在下一节中验证您的 Kubeflow 用户配置文件使用了正确的命名空间。

验证您的命名空间

这些步骤显示了如何验证您的活跃的 Kubeflow 用户配置文件是否使用了命名空间kubeflow-user-example-com。本系列的所有教程都在这个命名空间中运行。

1.

> **Note**
>
> 在 Kubeflow 中，所有命名空间都应通过配置文件创建。默认情况下，安装了AWS Vanilla 的 Kubeflow 会使用命kubeflow-user-example-com名空间创建用户配置文件。

通过运行以下命令确保名为的命名空间kubeflow-user-example-com存在。

```
kubectl get namespace
```

如果命名空间未出现在输出中，请按如下方式创建新的 Kubeflow 配置文件。

2. 打开vi或vim，然后复制并粘贴以下内容。将此配置文件描述文件另存为profile.yaml。确保将下面的电子邮件替换owner.name为您的电子邮件。

```
apiVersion: kubeflow.org/v1beta1
kind: Profile
metadata:
 # replace with the name of profile you want, this is the user's namespace name
 name: kubeflow-user-example-com
spec:
 owner:
     kind: User
     # replace with the email of the user
     name: user@example.com
```

3. 运行以下命令以创建相应的配置文件资源。

```
kubectl apply -f profile.yaml
```

4. 导出NAMESPACE变量。

```
export NAMESPACE=kubeflow-user-example-com
```

我们在所有 KubeflowAWS 教程${NAMESPACE}中将这个命名空间称为变量。

后续步骤

现在你已经完成了 Kubeflow 的AWS设置，你可以继续学习训练和推理教程。

要了解有关在 Kubeflow 上使用Deep Learning Containers 进行AWS训练和推理的信息，请参阅培训或推理指南。

清除

本节提供您完成教程运行后的清理说明。

清除工作

运行完示例后，您可以删除特定的训练作业。要列出在给定命名空间中运行的特定类型（PyTorchJobmpiJob， TfJob）的作业，请运行以下命令。

```
kubectl get job_type -n ${NAMESPACE}
```

检索要删除的任务的名称，然后运行以下命令。

```
kubectl delete job_type job_name -n ${NAMESPACE}
```

您的输出应类似于以下内容。

```
job_type.kubeflow.org "job_name" deleted
```

开启卸载 KubeflowAWS

AWS文档中的 Kubeflow 提供了卸载命令。确保运行与您的部署方法相对应的命令：Kustomize、Helm 或 Terrafor m。

删除 Amazon EKS 集群

AWS文档中的 Kubeflow 提供了一个命令来删除您的整个 Amazon EKS 集群。

目录

培训

在这些 KubeflowAWS 教程中，你可以学习 PyTorch 和使用Deep Learning Containers 和 Kubeflow 在 CPU 和 GPU 实例上AWS进行 TensorFlow 训练。

AWS安装时的 Kubeflow按照说明创建集群后，即可开始本次训练。

当您运行本系列中的一个 PyTorch 或 TensorFlow 训练示例时，请在与您的 Amazon EKS 集群匹配的类别中选择训练示例。这些类别是 CPU、GPU 或分布式 GPU。

CPU 训练

本节介绍如何使用 Kubeflow 训练运算符和Deep Learning Containers 在 CPU 实例上训练模型。

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像。有关使用英特尔数学内核库 (MKL) 时配置设置的提示，请参阅AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议。

目录

PyTorch 中央处理器训练

随之而AWS来的是你对 Kubeflow 的部署PyTorchJob。这是 Kub ernetes 自定义资源的 Kube flow 实现，用于在 Kubernetes 上运行分布式 PyTorch 训练作业。

本教程指导您在 MNIST 上训练分类模型，在单节点 CPU 实例 PyTorch中运行由 Kubeflow 管理的 Deep L earning Containers 中的容器AWS。

1. 要创建 PyTorchJob，请按照以下说明进行操作。

    1. 创建任务配置文件。

        打开vi或vim，然后复制并粘贴以下内容。将此文件另存为 pytorch.yaml。

```
apiVersion: "kubeflow.org/v1"
kind: PyTorchJob
metadata:
  name: pytorch-training
spec:
  pytorchReplicaSpecs:
    Worker:
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              imagePullPolicy: Always
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:2.0.0-cpu-py310-ubuntu20.04-ec2
```

```
            command:
              - "/bin/sh"
              - "-c"
            args:
            - "git clone https://github.com/pytorch/examples.git  && python
examples/mnist/main.py --no-cuda --epochs=1"
            env:
            - name: OMP_NUM_THREADS
              value: "36"
            - name: KMP_AFFINITY
              value: "granularity=fine,verbose,compact,1,0"
            - name: KMP_BLOCKTIME
              value: "1"
```

2. 使用部署 PyTorchJob 配置文件kubectl以开始训练。

```
kubectl create -f pytorch.yaml -n ${NAMESPACE}
```

该任务创建了一个运行来自Deep Learning Containers 容器的容器的 pod。这是在上述 YAML 文件容器名称下的字段spec.containers.image中引用的pytorch。

3. 您应当看到如下输出。

```
pytorchjob.kubeflow.org/pytorch-training created
```

4. 检查状态。

作业名称pytorch-training显示在状态中。这份工作可能需要一段时间才能到达一个Running州。运行以下命令watch command来监视作业的状态。

```
kubectl get pods -n ${NAMESPACE} -w
```

您应当看到如下输出。

```
NAME READY STATUS RESTARTS AGE
pytorch-training 0/1 Running 8 19m
```

2. 监控你的 PyTorchJob

1. 查看日志以查看训练进度。

```
kubectl logs pytorch-training-worker-0 -n ${NAMESPACE}
```

您应该可以看到类似于如下输出的内容。

```
Cloning into 'examples'...
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../
data/MNIST/raw/train-images-idx3-ubyte.gz
9920512it [00:00, 40133996.38it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../
data/MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
32768it [00:00, 831315.84it/s]
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../
data/MNIST/raw/t10k-images-idx3-ubyte.gz
1654784it [00:00, 13019129.43it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../
data/MNIST/raw/t10k-labels-idx1-ubyte.gz
8192it [00:00, 337197.38it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
Processing...
Done!
Train Epoch: 1 [0/60000 (0%)]    Loss: 2.300039
Train Epoch: 1 [640/60000 (1%)]    Loss: 2.213470
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.170460
Train Epoch: 1 [1920/60000 (3%)]    Loss: 2.076699
Train Epoch: 1 [2560/60000 (4%)]    Loss: 1.868078
Train Epoch: 1 [3200/60000 (5%)]    Loss: 1.414199
Train Epoch: 1 [3840/60000 (6%)]    Loss: 1.000870
```

2. 监控作业状态。

   运行以下命令来刷新任务状态。当状态更改为时Succeeded，训练作业即告完成。

```
watch -n 5 kubectl get pytorchjobs pytorch-training -n ${NAMESPACE}
```

清除有关在使用完集群后对其进行清理的信息，请参阅。

TensorFlow中央处理器训练

你在 Kubeflow 上部署 TFJobAWS 是随附的。这是 Kub ernetes 自定义资源的 Kube flow 实现，用于在 Kubernetes 上运行分布式 TensorFlow 训练作业。

本教程指导你在运行由 Kubeflow 管理的Deep Learning Containers 中的容器的单节点 CPU 实例中使用 Keras 在 MNIST 上训练分类模型AWS。

1. 创建一个 TFJob。

   1. 创建任务配置文件。

      打开vi或vim，然后复制并粘贴以下内容。将此文件另存为 tf.yaml。

```
apiVersion: kubeflow.org/v1
kind: TFJob
metadata:
  name: tensorflow-training
spec:
  tfReplicaSpecs:
    Worker:
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: tensorflow
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
training:2.12.0-cpu-py310-ubuntu20.04-ec2
              command: ["/bin/sh","-c"]
              args: ["git clone https://github.com/keras-team/keras-io.git &&
 python keras-io/examples/vision/mnist_convnet.py"]
```

   2. 要开始训练，请使用部署 TFJob 配置文件kubectl。

```
kubectl create -f tf.yaml -n ${NAMESPACE}
```

      该任务通过运行您在上述 YAML 文件字段spec.containers.image中的容器名称下引用的 Deep Learning Containers 来创建 Podtensorflow。

3. 您应当看到如下输出。

```
pod/tensorflow-training created
```

4. 检查状态。

   作业名称tensorflow-training显示在状态中。这份工作可能需要一段时间才能到达一个Running州。运行以下 watch 命令来监视作业的状态。

```
kubectl get pods -n ${NAMESPACE} -w
```

   您应当看到如下输出。

```
NAME READY STATUS RESTARTS AGE
tensorflow-training 0/1 Running 8 19m
```

2. 监控您的 Tob。

1. 查看日志以查看训练进度。

```
kubectl logs tensorflow-training-worker-0 -n ${NAMESPACE}
```

   您应该可以看到类似于如下输出的内容。

```
Cloning into 'keras'...
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz

    8192/11490434 [..............................] - ETA: 0s
 6479872/11490434 [===============>..............] - ETA: 0s
 8740864/11490434 [=====================>........] - ETA: 0s
11493376/11490434 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
2019-03-19 01:52:33.863598: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
 use: AVX512F
```

```
2019-03-19 01:52:33.867616: I tensorflow/core/common_runtime/process_util.cc:69]
 Creating new thread pool with default inter op setting: 2. Tune using
 inter_op_parallelism_threads for best performance.

  128/60000 [..............................] - ETA: 10:43 - loss: 2.3076 - acc:
0.0625
  256/60000 [..............................] - ETA: 5:59 - loss: 2.2528 - acc:
0.1445
  384/60000 [..............................] - ETA: 4:24 - loss: 2.2183 - acc:
0.1875
  512/60000 [..............................] - ETA: 3:35 - loss: 2.1652 - acc:
0.1953
  640/60000 [..............................] - ETA: 3:05 - loss: 2.1078 - acc:
0.2422
  ...
```

2. 监控作业状态。

   运行以下命令来刷新任务状态。当状态更改为时Succeeded，训练作业即告完成。

   ```
   watch -n 5 kubectl get tfjobs tensorflow-training -n ${NAMESPACE}
   ```

清除有关在使用完集群后对其进行清理的信息，请参阅。

后续步骤

要在AWS使用 PyTorch 或 TensorFlow 使用Deep Learning Containers 时在 Kubeflow 上学习基于 CPU 的推理，请参阅推理。

GPU 训练

本节演示如何使用 Kubeflow 训练运算符和Deep Learning Containers 在 GPU 实例上训练模型。

运行示例之前，请确保您的集群有 GU 节点。如果您的集群中没有 GPU 节点，请使用以下命令向集群添加节点组。请务必在 “加速计算” 类别中选择 Amazon EC2 实例 (node-type)。

```
eksctl create nodegroup --cluster $CLUSTER_NAME --region $CLUSTER_REGION \
      --nodes 2 --nodes-min 1 --nodes-max 3 --node-type p3.2xlarge
```

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像。有关使用英特尔数学内核库 (MKL) 时配置设置的提示，请参阅AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议。

目录

PyTorch GPU 训练

随之而AWS来的是你对 Kubeflow 的部署PyTorchJob。这是 Kub ernetes 自定义资源的 Kube flow 实现，用于在亚马逊 EKS 上运行分布式 PyTorch 训练任务。

本教程展示了如何在单节点 GPU 实例 PyTorch 中训练模型。你将在你的容器中运行这个 PyTorch MNIST 示例，该Deep Learning Containers 由 Kubeflow 管理AWS。

1. 创建一个 PyTorchJob.

    1. 创建任务配置文件。

        打开vi或vim，然后复制并粘贴以下内容。将此文件另存为 pytorch.yaml。

```
apiVersion: "kubeflow.org/v1"
kind: PyTorchJob
metadata:
  name: pytorch-training
spec:
  pytorchReplicaSpecs:
    Worker:
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              imagePullPolicy: Always
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:2.0.0-gpu-py310-cu118-ubuntu20.04-ec2
              command:
                - "/bin/sh"
                - "-c"
              args:
              - "git clone https://github.com/pytorch/examples.git && python
 examples/mnist/main.py --no-cuda --epochs=1"
```

```
            env:
            - name: OMP_NUM_THREADS
              value: "36"
            - name: KMP_AFFINITY
              value: "granularity=fine,verbose,compact,1,0"
            - name: KMP_BLOCKTIME
              value: "1"
            resources:
              limits:
                nvidia.com/gpu: 1
```

2. 使用部署 PyTorchJob 配置文件kubectl以开始训练。

```
kubectl create -f pytorch.yaml -n ${NAMESPACE}
```

该任务创建了一个运行该容器的 Pod，该容器来自字段中引用的Deep Learning Containerspec.containers.image s。它位于上面的容器名称下的 YAML 文件中pytorch。

3. 您应当看到如下输出。

```
pod/pytorch-training created
```

4. 检查状态。

作业名称pytorch-training显示在状态中。这份工作可能需要一段时间才能到达一个Running州。运行以下 watch 命令来监视作业的状态。

```
kubectl get pods n ${NAMESPACE} -w
```

您应当看到如下输出。

```
NAME READY STATUS RESTARTS AGE
pytorch-training 0/1 Running 8 19m
```

2. 监视你的 PyTorchJob.

1. 查看日志以查看训练进度。

```
kubectl logs pytorch-training-worker-0 -n ${NAMESPACE}
```

您应该可以看到类似于如下输出的内容。

```
Cloning into 'examples'...
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../
data/MNIST/raw/train-images-idx3-ubyte.gz
9920512it [00:00, 40133996.38it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../
data/MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
32768it [00:00, 831315.84it/s]
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../
data/MNIST/raw/t10k-images-idx3-ubyte.gz
1654784it [00:00, 13019129.43it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../
data/MNIST/raw/t10k-labels-idx1-ubyte.gz
8192it [00:00, 337197.38it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
Processing...
Done!
Train Epoch: 1 [0/60000 (0%)]    Loss: 2.300039
Train Epoch: 1 [640/60000 (1%)]    Loss: 2.213470
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.170460
Train Epoch: 1 [1920/60000 (3%)]    Loss: 2.076699
Train Epoch: 1 [2560/60000 (4%)]    Loss: 1.868078
Train Epoch: 1 [3200/60000 (5%)]    Loss: 1.414199
Train Epoch: 1 [3840/60000 (6%)]    Loss: 1.000870
```

2. 监控作业状态。

   运行以下命令来刷新任务状态。当状态更改为时Succeeded，训练作业即告完成。

   ```
   watch -n 5 kubectl get pytorchjobs pytorch-training -n ${NAMESPACE}
   ```

清除有关在使用完集群后对其进行清理的信息，请参阅。

TensorFlow GPU 训练

你在 Kubeflow 上部署 TFJobAWS 是随附的。这是 Kub ernetes 自定义资源的 Kube flow 实现，用于在 Kubernetes 上运行分布式 TensorFlow 训练作业。

本教程指导您在单节点 GPU 实例中使用 Keras 在 MNIST 上训练分类模型，该实例运行由 Kubeflow 管理的Deep Learning Containers 中的容器AWS。

1. 创建一个 TFJob。

   1. 创建任务配置文件。

      打开vi或vim，然后复制并粘贴以下内容。将此文件另存为 tf.yaml。

```
apiVersion: kubeflow.org/v1
kind: TFJob
metadata:
  name: tensorflow-training
spec:
  tfReplicaSpecs:
    Worker:
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: tensorflow
              image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
training:2.12.0-gpu-py310-cu118-ubuntu20.04-ec2
              command: ["/bin/sh","-c"]
              args: ["git clone https://github.com/keras-team/keras-io.git &&
 python keras-io/examples/vision/mnist_convnet.py"]
              resources:
                limits:
                  nvidia.com/gpu: 1
```

   2. 使用部署 TFJob 配置文件kubectl以开始训练。

```
kubectl create -f tf.yaml ${NAMESPACE}
```

      该任务创建了一个运行该容器的 Pod，该容器来自字段中引用的Deep Learning Containerspec.containers.image s。它位于上面的容器名称下的 YAML 文件中tensorflow。

   3. 您应当看到如下输出。

```
pod/tensorflow-training created
```

4. 检查状态。

   作业名称tensorflow-training显示在状态中。这份工作可能需要一段时间才能到达一个Running 州。运行以下 watch 命令来监视作业的状态。

   ```
   watch -n 5 kubectl get pods ${NAMESPACE}
   ```

   您应当看到如下输出。

   ```
   NAME READY STATUS RESTARTS AGE
   tensorflow-training 0/1 Running 8 19m
   ```

2. 监控您的 Tob。

   1. 查看日志以查看训练进度。

      ```
      kubectl logs tensorflow-training-worker-0 ${NAMESPACE}
      ```

      您应该可以看到类似于如下输出的内容。

      ```
      Cloning into 'keras'...
      Using TensorFlow backend.
      Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz

          8192/11490434 [..............................] - ETA: 0s
       6479872/11490434 [===============>..............] - ETA: 0s
       8740864/11490434 [=====================>........] - ETA: 0s
      11493376/11490434 [==============================] - 0s 0us/step
      x_train shape: (60000, 28, 28, 1)
      60000 train samples
      10000 test samples
      Train on 60000 samples, validate on 10000 samples
      Epoch 1/12
      2019-03-19 01:52:33.863598: I tensorflow/core/platform/cpu_feature_guard.cc:141]
       Your CPU supports instructions that this TensorFlow binary was not compiled to
       use: AVX512F
      ```

```
2019-03-19 01:52:33.867616: I tensorflow/core/common_runtime/process_util.cc:69]
 Creating new thread pool with default inter op setting: 2. Tune using
 inter_op_parallelism_threads for best performance.

  128/60000 [..............................] - ETA: 10:43 - loss: 2.3076 - acc:
0.0625
  256/60000 [..............................] - ETA: 5:59 - loss: 2.2528 - acc:
0.1445
  384/60000 [..............................] - ETA: 4:24 - loss: 2.2183 - acc:
0.1875
  512/60000 [..............................] - ETA: 3:35 - loss: 2.1652 - acc:
0.1953
  640/60000 [..............................] - ETA: 3:05 - loss: 2.1078 - acc:
0.2422
  ...
```

2. 监控作业状态。

   运行以下命令来刷新任务状态。当状态更改为时Succeeded，训练作业即告完成。

   ```
   watch -n 5 kubectl get tfjobs tensorflow-training ${NAMESPACE}
   ```

清除有关在使用完集群后对其进行清理的信息，请参阅。

分布式 GPU 训练

本节适用于基于 GPU 的集群上的分布式训练。

运行示例之前，请确保您的集群有 GU 节点。如果您的集群中没有 GPU 节点，请使用以下命令向集群添加节点组。请务必在 “加速计算” 类别中选择 Amazon EC2 实例 (node-type)。

```
eksctl create nodegroup --cluster $CLUSTER_NAME --region $CLUSTER_REGION \
      --nodes 2 --nodes-min 1 --nodes-max 3 --node-type p3.2xlarge
```

有关Deep Learning Containers 完整列表，请参阅Deep Learning Containers 映像。

目录

PyTorch分布式 GPU 训练

本教程指导您在 MNIST 上训练分类模型，在单节点 GPU 实例 PyTorch中运行由 Kubeflow 管理的 Deep L earning Containers 中的容器AWS。该示例使用 Gloo 作为后端。

1. 创建一个 PyTorchJob.

    1. 验证 PyTorch 自定义资源是否已安装。

        ```
        kubectl get crd
        ```

        该输出应包含 pytorchjobs.kubeflow.org。

    2. 确保 NVIDIA 插件daemonset正在运行。

        ```
        kubectl get daemonset -n kube-system
        ```

        该输出值应该类似于以下内容。

        ```
        NDESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
        nvidia-device-plugin-daemonset   3         3      3        3        3    <none>
         35h
        ```

    3. 使用以下文本创建基于 Gloo 的分布式数据并行作业。将其保存在名为的文件中pt_distributed.yaml。

        ```
        apiVersion: kubeflow.org/v1
        kind: PyTorchJob
        metadata:
          name: "kubeflow-pytorch-gpu-dist-job"
        spec:
          pytorchReplicaSpecs:
            Master:
              replicas: 1
              restartPolicy: OnFailure
              template:
                metadata:
                  annotations:
                    sidecar.istio.io/inject: "false"
                spec:
                  containers:
        ```

```
            - name: "pytorch"
              image: "763104351884.dkr.ecr.us-west-2.amazonaws.com/aws-samples-
pytorch-training:2.0-gpu-py310-ec2"
              args:
                - "--backend"
                - "gloo"
                - "--epochs"
                - "5"
    Worker:
      replicas: 2
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-west-2.amazonaws.com/aws-samples-
pytorch-training:2.0-gpu-py310-ec2"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
            resources:
              limits:
                nvidia.com/gpu: 1
```

4. 运行分布式训练的训练分布式训练的训练的训练

```
kubectl create -f pt_distributed.yaml -n ${NAMESPACE}
```

2. 监视你的 PyTorchJob.

1. 查看状态部分以监控作业状态。以下是任务成功完成时的输出示例。

```
kubectl get -o yaml pytorchjobs kubeflow-pytorch-gpu-dist-job ${NAMESPACE}
```

2. 检查每个 pod 的日志。

   第一个命令来打印出特定的 Pod 列表 PyTorchJob，如以下示例所示。

```
kubectl get pods -l job-name=kubeflow-pytorch-gpu-dist-job -o name -n
 ${NAMESPACE}
```

第二条命令跟踪特定 pod 的日志。

```
kubectl logs pod name -n ${NAMESPACE}
```

有关清除在使用完集群后对其进行清理的信息，请参阅。

TensorFlow 使用 Horovod 分布式 GPU 训练

本教程将指导您在 GPU 集群 TensorFlow 上使用 Horovod 完成分布式训练。你将在你的容器 TensorFlow中运行这个分布式训练示例， ResNet 基于 Kubeflow on 管理的Deep ImageNet L earning ContainersAWS。

该示例要求一个 GPU 实例至少有 2 个 GPU。你可以使用node-type=p3.16xlarge或更高版本。

1. 创建 mpiJob。

   1. 验证 TensorFlow 自定义资源是否已安装。

      ```
      kubectl get crd
      ```

      该输出应包含 mpijobs.kubeflow.org。

   2. 确保 NVIDIA 插件daemonset正在运行。

      ```
      kubectl get daemonset -n kube-system
      ```

      该输出值应该类似于以下内容。

      ```
      NDESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
      nvidia-device-plugin-daemonset   3         3      3        3        3    <none>
       35h
      ```

   3. 使用以下文本来创建 miJob。将其保存在名为的文件中tf_distributed.yaml.。

      ```
      apiVersion: kubeflow.org/v1
      ```

```
kind: MPIJob
metadata:
  name: tensorflow-tf-dist
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         metadata:
           annotations:
             sidecar.istio.io/inject: "false"
         spec:
           containers:
           - image: 763104351884.dkr.ecr.us-west-2.amazonaws.com/aws-samples-
tensorflow-training:2.12-gpu-py310-ec2
             name: tensorflow-launcher
             command:
              - mpirun
              - -mca
              - btl_tcp_if_exclude
              - lo
              - -mca
              - pml
              - ob1
              - -mca
              - btl
              - ^openib
              - --bind-to
              - none
              - -map-by
              - slot
              - -x
              - LD_LIBRARY_PATH
              - -x
              - PATH
              - -x
              - NCCL_SOCKET_IFNAME=eth0
              - -x
              - NCCL_DEBUG=INFO
              - -x
              - MXNET_CUDNN_AUTOTUNE_DEFAULT=0
              - python
```

```
              - /deep-learning-models/models/resnet/tensorflow2/
train_tf2_resnet.py
             args:
              - --num_epochs
              - "10"
              - --synthetic
    Worker:
      replicas: 2
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
          - image: 763104351884.dkr.ecr.us-west-2.amazonaws.com/aws-samples-
tensorflow-training:2.12-gpu-py310-ec2
            name: tensorflow-worker
            resources:
              limits:
                nvidia.com/gpu: 1
```

4. 运行分布式训练的训练分布式训练的训练的训练

```
kubectl create -f tf_distributed.yaml -n ${NAMESPACE}
```

2. 监视你的 PyTorchJob.

1. 查看状态部分以监控作业状态。以下是任务成功完成后的输出示例。

```
kubectl get -o yaml mpijob tensorflow-tf-dist -n ${NAMESPACE}
```

2. 检查每个 pod 的日志。

   第一个命令打印特定的 pod 列表 PyTorchJob，例如以下示例。

```
kubectl get -o yaml mpijob tensorflow-tf-dist -n ${NAMESPACE}
```

   第二条命令跟踪特定 pod 的日志。

```
kubectl logs pod name -n ${NAMESPACE}
```

有关清除在使用完集群后对其进行清理的信息，请参阅。

推理

本指南介绍如何在 PyTorch 或 TensorFlow 模型上运行推理服务。

如果您已经创建了集群并在上部署 KubeflowAWS，则可以开始本教程。如果不是，请按照中的步骤操作AWS安装时的 Kubeflow。根据您的集群设置选择 CPU 或 GPU 示例。推理示例在单节点配置上运行。

TensorFlow使用 kServe 进行CPU推断

KServe 在 Kubernetes 上为常见的机器学习 (ML) 框架启用无服务器推理。框架包括 TensorFlow xgBoost 或 PyTorch。kServe 是在开启 Kubeflow 的情况下预安装的AWS。在本教程中，您将创建一个 kServe 服务来在 CPU 集群上运行 TensorFlow 模型推断。

> **Note**
>
> 在本示例中，该服务在集群内部 IP 上公开`ClusterIP`。在生产环境中，您可能需要使用负载均衡器在外部公开推理服务。

1. 在 Kubeflow 1.7 中，默认情况下，推理服务不通过 kubeflow-gateway 配置外部 DNS。要解决此问题，请运行以下命令，除非您已经配置了自定义域。有关更多详细信息，请关注此GitHub 问题。

   ```
   kubectl patch cm config-domain --patch '{"data":{"example.com":""}}' -n knative-serving
   ```

2. 利用以下内容创建`tf_inference.yaml`名为。此示例指定了我们的推理服务使用的模型和TensorFlow推理图像的远程位置。该模型是 kServe 提供的公开示例，无需修改即可使用。

   ```
   apiVersion: "serving.kserve.io/v1beta1"
   kind: "InferenceService"
   metadata:
     name: "flower-sample"
     annotations:
       sidecar.istio.io/inject: "false"
   spec:
     predictor:
       tensorflow:
   ```

```
      image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:2.11.1-cpu-py39-ubuntu20.04-ec2"
      storageUri: "gs://kfserving-examples/models/tensorflow/flowers"
```

3. 将服务描述应用于集群中的 pod。

```
kubectl apply -f tf_inference.yaml -n ${NAMESPACE}
```

如下所示。

```
inferenceservice.serving.kserve.io/flower-sample created
```

4. READY通过运行以下命令检查推理服务的状态以确保其处于状态。推理服务可能需要几分钟才能启动。

```
kubectl get inferenceservices flower-sample -n ${NAMESPACE}
```

在命令输出中，部署状态应true在READY列下方。

```
NAME            URL                                                    READY
  PREV   LATEST   PREVROLLEDOUTREVISION   LATESTREADYREVISION
 AGE
flower-sample   http://flower-sample.kubeflow-user-example-com.example.com  True
         100                              flower-sample-predictor-default-00001
 3m31s
```

5. 使用以下命令查看集群状态。

```
kubectl get pods -n ${NAMESPACE}
```

通过检查命令输出来确认 podRunning 处于状态。STATUS

```
NAME                                                        READY   STATUS
 RESTARTS   AGE
flower-sample-predictor-default-00001-deployment-76c89dc6c47cvl   3/3     Running
 0          24s
```

6. 要进一步描述 pod，请运行以下命令。

```
kubectl describe pod pod_name -n ${NAMESPACE}
```

7. 要访问推理服务，请将端口从容器转发到主机。在典型的推理服务部署中，您很可能需要使用负载均衡器设置更永久的解决方案。此命令在终端的前台持续运行。

```
INGRESS_GATEWAY_SERVICE=$(kubectl get svc --namespace istio-system --
selector="app=istio-ingressgateway" --output jsonpath='{.items[0].metadata.name}')
kubectl port-forward --namespace istio-system svc/${INGRESS_GATEWAY_SERVICE}
 8080:80
```

8. 使用此命令下载输入样本数据。该命令创建一个flower_input.json包含示例数据的文件。

```
curl https://raw.githubusercontent.com/kserve/kserve/release-0.8/docs/samples/
v1beta1/tensorflow/input.json -o flower_input.json
```

9. 在单独的终端中，通过创建和运行以下脚本登录推理服务。

   a. 打开vi或vim，然后将以下脚本复制并粘贴到名为的文件中inference_authentication.py。该脚本触发 OpenID Connect (OIDC) 交换，从而生成会话 cookie 来验证推理请求。

```
import requests
import os
import json

CLUSTER_IP = os.environ.get("CLUSTER_IP", "localhost:8080")
DASHBOARD_URL = f"http://{CLUSTER_IP}"
NAMESPACE = os.environ.get("NAMESPACE", "kubeflow-user-example-com")
MODEL_NAME = os.environ.get("MODEL_NAME", "sklearn-iris")
SERVICE_HOSTNAME = os.environ.get("SERVICE_HOSTNAME", "flower-sample.kubeflow-
user-example-com.example.com")
URL = f"http://{CLUSTER_IP}/v1/models/{MODEL_NAME}:predict"
HEADERS = {"Host": f"{SERVICE_HOSTNAME}"}
USERNAME = os.environ.get("USERNAME", "user@example.com")
PASSWORD = os.environ.get("PASSWORD", "12341234")

def load_json_file(filepath):
    with open(filepath) as file:
        return json.load(file)

data = load_json_file("./flower_input.json")

response = None
```

```
def session_cookie(host, login, password):
    session = requests.Session()
    response = session.get(host)
    headers = {
        "Content-Type": "application/x-www-form-urlencoded",
    }
    data = {"login": login, "password": password}
    session.post(response.url, headers=headers, data=data)
    session_cookie = session.cookies.get_dict()["authservice_session"]
    return session_cookie

cookie = {"authservice_session": session_cookie(DASHBOARD_URL, USERNAME,
 PASSWORD)}
response = requests.post(URL, headers=HEADERS, json=data, cookies=cookie)

print("Sending request to:", URL)

status_code = response.status_code
print("Status Code", status_code)
if status_code == 200:
    print("JSON Response ", json.dumps(response.json(), indent=2))
```

b. 要使用示例输入数据运行预测，请使用以下命令运行上面的脚本。

```
export INGRESS_HOST=localhost
export INGRESS_PORT=8080
export CLUSTER_IP=${INGRESS_HOST}:${INGRESS_PORT}
export NAMESPACE=kubeflow-user-example-com
export MODEL_NAME=flower-sample
export SERVICE_HOSTNAME=$(kubectl get -n ${NAMESPACE} inferenceservice
 ${MODEL_NAME} -o jsonpath='{.status.url}' | cut -d "/" -f 3)
export USERNAME=user@example.com
export PASSWORD=12341234
```

```
pip install requests
```

```
python3 ./inference_authentication.py
```

10. 如下所示。

```
Sending request to: http://localhost:8080/v1/models/flower-sample:predict
Status Code 200
```

```
JSON Response  {
  "predictions": [
    {
      "prediction": 0,
      "key": "   1",
      "scores": [
        0.999114931,
        9.20989623e-05,
        0.000136786606,
        0.000337258854,
        0.000300534302,
        1.84814289e-05
      ]
    }
  ]
}
```

清除有关在使用完集群后对其进行清理的信息，请参阅。

自定义入口点

对于一些图片，AWS容器使用自定义入口点脚本。如果您要使用自己的入口点，可以按如下方式覆盖入口点。

更新 Pod 文件中的 command 参数。将 args 参数替换为您的自定义入口点脚本。

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-multi-model-server-densenet
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-multi-model-server-densenet
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.2.0-cpu-
py36-ubuntu16.04
    command:
      - "/bin/sh"
      - "-c"
    args:
      - /usr/local/bin/mxnet-model-server
```

```
      - --start
      - --mms-config /home/model-server/config.properties
      - --models densenet="https://dlc-samples.s3.amazonaws.com/pytorch/multi-model-
server/densenet/densenet.mar"
```

command 是 entrypoint 的 Kubernetes 字段名称。有关详细信息，请参阅此 Kubernetes 字段名称表。

如果 EKS 集群具有过期的 IAM 权限访问包含该映像的 ECR 存储库，或者您使用的 kubectl 来自与创建该集群的用户不同的用户，则您将收到以下错误消息。

```
error: unable to recognize "job.yaml": Unauthorized
```

要解决此问题，您需要刷新 IAM 令牌。请运行以下脚本。

```
set -ex

AWS_ACCOUNT=${AWS_ACCOUNT}
AWS_REGION=us-east-1
DOCKER_REGISTRY_SERVER=https://${AWS_ACCOUNT}.dkr.ecr.${AWS_REGION}.amazonaws.com
DOCKER_USER=AWS
DOCKER_PASSWORD=`aws ecr get-login --region ${AWS_REGION} --registry-ids ${AWS_ACCOUNT}
 | cut -d' ' -f6`
kubectl delete secret aws-registry || true
kubectl create secret docker-registry aws-registry \
--docker-server=$DOCKER_REGISTRY_SERVER \
--docker-username=$DOCKER_USER \
--docker-password=$DOCKER_PASSWORD
kubectl patch serviceaccount default -p '{"imagePullSecrets":[{"name":"aws-
registry"}]}'
```

将 spec 下的以下内容附加到您的 Pod 文件中。

```
imagePullSecrets:
    - name: aws-registry
```

故障排除AWSEKS 上的 Deep Learning Containers

以下是使用时可能会在命令行中返回的常见错误AWSAmazon EKS 群集上的 Deep Learning Containers。每个错误之后都是错误的解决方案。

Troubleshooting

主题

设置错误

在 Amazon EKS 群集上设置 Deep Learning Containers 时，可能会返回以下错误。

- 错误：注册表**kubeflow**不存在

```
$ ks pkg install kubeflow/tf-serving
  ERROR registry 'kubeflow' does not exist
```

要解决此错误，请运行以下命令。

```
ks registry add kubeflow github.com/google/kubefl ow/tree/master/kubeflow
```

- 错误：上下文超出截止日

```
$ eksctl create cluster <args>
  [#] waiting for CloudFormation stack "eksctl-training-cluster-1-nodegroup-
ng-8c4c94bc" to reach "CREATE_COMPLETE" status: RequestCanceled: waiter context
 canceled
  caused by: context deadline exceeded
```

要解决此错误，请验证是否超出您的账户的容量。你也可以尝试在其他区域创建集群。

- Error: 与服务器本地主机:8080 的连接被拒绝

```
$ kubectl get nodes
  The connection to the server localhost:8080 was refused - did you specify the right
 host or port?
```

要解决此错误，请运行以下命令将集群复制到 Kubernetes 配置。

```
cp ~/.kube/eksctl/clusters/<cluster-name> ~/.kube/config
```

- 错误：处理对象：从群集中修补对象：将对象与现有状态合并：未授权

```
$ ks apply default
  ERROR handle object: patching object from cluster: merging object with existing
 state: Unauthorized
```

此错误是由于并发问题造成的，当具有不同授权或凭证凭证的多个用户试图在同一集群上启动作业时，便会发生此问题。验证您是否在正确的群集上启动作业。

- Error: 无法创建应用程序；目录 '/家/ubuntu/kubeflow-tf-hvd' 已存在

```
$ APP_NAME=kubeflow-tf-hvd; ks init ${APP_NAME}; cd ${APP_NAME}
  INFO Using context "arn:aws:eks:eu-west-1:999999999999:cluster/training-gpu-1" from
 kubeconfig file "/home/ubuntu/.kube/config"
  ERROR Could not create app; directory '/home/ubuntu/kubeflow-tf-hvd' already exists
```

您可以放心地忽略此警告。但是，您可能需要对该文件夹执行额外的清理工作。要简化清理工作，请删除文件夹。

使用错误

```
ssh: Could not resolve hostname openmpi-worker-1.openmpi.kubeflow-dist-train-tf: Name
 or service not known
```

如果您在使用 Amazon EKS 群集时看到此错误消息，则针对此错误消息再运行一遍 NVIDIA 设备插件安装步骤。通过以下方式验证针对的是正确的集群：传入特定配置文件或切换活动集群到目标集群。

清理错误

清理 Amazon EKS 群集的资源时，可能会返回以下错误。

- 错误：服务器没有资源类型”*namspace*"

```
$ kubectl delete namespace ${NAMESPACE}
  error: the server doesn't have a resource type "namspace"
```

验证命名空间的拼写是否正确。

- 错误：服务器已要求客户端提供凭据

```
$ ks delete default
  ERROR the server has asked for the client to provide credentials
```

要解决此错误，请验证~/.kube/config指向正确的集群那AWS已使用正确配置凭据aws configure或者通过出口AWS环境变量。

- 错误：从起始路径找到应用程序根目录# 找不到 ksonnet 项目

```
$ ks delete default
  ERROR finding app root from starting path: : unable to find ksonnet project
```

要解决此错误，请验证您是否位于 ksonnet 应用程序创建的目录中。这是在哪里的文件夹ks init运行了。

- Error: 来自服务器的错误（未找到）：找不到 pod “openmpi-master”

```
$ kubectl logs -n ${NAMESPACE} -f ${COMPONENT}-master > results/benchmark_1.out
  Error from server (NotFound): pods "openmpi-master" not found
```

此错误可能是由于在删除上下文后尝试访问资源引起的。删除默认上下文也会导致相应的资源也被删除。

框架Support 政策

AWSD@@ eep Learning Containers (DLC) 简化了深度学习工作负载的映像配置，并使用最新的框架、硬件、驱动程序、库和操作系统进行了优化。本页详细介绍了 DLC 的框架支持政策。有关可用 DLC 的列表，请参阅De ep Learning Containers 发行说明。

支持的框架

参考下面的 AWSDeep Learning Containers 框架Support 策略表，查看哪些框架和版本得到积极支持。

请参阅 E nd of patch 查看原始框架维护团队积极支持的当前版本AWS支持多长时间。框架和版本可在单框架 DLC 中找到。

> **Note**
>
> 在框架版本 x.y.z 中，x 指主要版本，y 指次要版本，z 指补丁版本。例如，对于 TensorFlow 2.6.5，主要版本为 2，次要版本为 6，补丁版本为 5。

有关特定图像的更多详细信息，请参阅发行说明：

- 单框架 DLC 发行说明
- 可用的Deep Learning Containers 图片页面

常见问题

- 哪些框架版本会获得安全补丁？
- 新框架版本AWS发布时会发布哪些图像？
- 哪些图像有新增 SageMaker/AWS功能？
- 当前版本在支持的框架表中是如何定义的？
- 如果我运行的版本不在 “支持的框架” 表中，该怎么办？
- DLC 是否支持以前版本的 TensorFlow？
- 我怎样才能找到支持的框架版本的最新补丁镜像？
- 新图片的发布频率如何？

哪些框架版本会获得安全补丁？

如果框架版本在 “Dee AWSp Learning Containers 框架Support 策略” 表中标记为 “支持”，则会获得安全补丁。

新框架版本AWS发布时会发布哪些图像？

我们会在和的新版本发布后不久发布新 TensorFlow PyTorch的 DLC。这包括主要版本、主要次要版本和框架 major-minor-patch 版本。当新版本的驱动程序和库可用时，我们还会更新映像。有关映像维护的更多信息，请参阅对我的框架版本的主动支持何时结束？

哪些图像有新增 SageMaker/AWS功能？

新功能通常会在 PyTorch 和的最新版本的 DLC 中发布 TensorFlow。有关新增 SageMaker 或AWS 功能的详细信息，请参阅特定图像的发行说明。有关可用 DLC 的列表，请参阅De AWSep Learning Containers 发行说明。有关映像维护的更多信息，请参阅对我的框架版本的主动支持何时结束？

当前版本在支持的框架表中是如何定义的？

D AWSeep LAWS earning Containers 框架Support 策略表中的当前版本是指在上提供的最新框架版本 GitHub。每个最新版本都包含对 DLC 中驱动程序、库和相关软件包的更新。有关映像维护的信息，请参见对我的框架版本的主动支持何时结束？

如果我运行的版本不在 “支持的框架” 表中，该怎么办？

如果您运行的版本不在 D AWSeep Learning Containers Framework Support Policy 表中，则可能没有最新的驱动程序、库和相关软件包。要获得更多 up-to-date 版本，我们建议您使用您选择的最新 DLC

升级到可用的支持框架之一。有关可用 DLC 的列表，请参阅De AWSep Learning Containers 发行说明。

DLC 是否支持以前版本的 TensorFlow？

否。 如Dee AWSp Learning Containers 框架Support 政策表中所述，我们支持每个框架自首次 GitHub 发布之日起 365 天发布的最新主要版本的最新补丁版本。有关更多信息，请参阅如果我运行的版本不在 “支持的框架” 表中，该怎么办？

我怎样才能找到支持的框架版本的最新补丁镜像？

要使用具有最新框架版本的 DLC，请浏览 DLC GitHub 发行标签以找到您选择的示例图片 URI，然后使用它提取最新的可用的 Docker 镜像。您选择的框架版本必须在 “Dee AWSp Learning Containers 框架Support 策略” 表中标记为 “支持”。

新图片的发布频率如何？

提供更新的补丁版本是我们的首要任务。我们通常会尽早创建补丁图像。我们会监控新修补的框架版本（例如 TensorFlow 2.9 到 TensorFlow 2.9.1）和新的次要发行版本（例如 TensorFlow 2.9 到 TensorFlow 2.10），并尽早提供它们。当使用新版本 TensorFlow 的 CUDA 发布现有版本时，我们会为该版本发布新的 DLC， TensorFlow 并支持新的 CUDA 版本。

我的工作负载正在运行时，我的实例会被修补到位吗？

否。 DLC 的补丁更新不是 “就地” 更新。

您必须在不终止实例的情况下删除实例上的现有镜像并提取最新的容器镜像。

当有新的补丁或更新的框架版本可用时会发生什么？

定期在发行说明页面上查看您的图片。我们鼓励您在新的补丁或更新的框架可用时将其升级。有关可用 DLC 的列表，请参阅De AWSep Learning Containers 发行说明。

依赖关系是否在不更改框架版本的情况下进行了更新？

我们在不更改框架版本的情况下更新依赖关系。但是，如果依赖项更新导致不兼容，我们会使用不同的版本创建镜像。请务必查看D AWSeep Learning Containers 发行说明，了解更新的依赖关系信息。

对我的框架版本的主动支持何时结束？

DLC 图像是不可变的。它们一旦被创建，它们就不会改变。终止对框架版本的主动支持有四个主要原因：

- 框架版本（补丁）升级
- AWS安全补丁
- 补丁结束日期（过期）
- 依赖关系 end-of-support

> **Note**
>
> 由于版本补丁升级和安全补丁的频率很高，我们建议您经常查看 DLC 的发行说明页面，并在做出更改时进行升级。

框架版本（补丁）升级

如果您的 DLC 工作负载基于 TensorFlow 2.7.0 且 TensorFlow版本为 2.7.1 GitHub，则使用 TensorFlow 2.7.1AWS 发布新的 DLC。2.7.0 版本的新映像发布后，2.7.0 版本的先前映像将不再处于活跃状态。 TensorFlow TensorFlow 2.7.0 版本的 DLC 不会收到更多补丁。然后，使用最新信息更新 TensorFlow 2.7 的 DLC 发行说明页面。每个次要补丁都没有单独的发行说明页面。

由于补丁升级而创建的新 DLC 将使用更新的发布标签进行指定。如果更改不向后兼容，则标签将更改主要版本而不是次要版本（例如 v1.0 将更改为 v2.0 而不是 v 1.2）。

AWS安全补丁

如果您的工作负载基于 TensorFlow 2.7.0 版本的映像并AWS制作了安全补丁，则将发布适用于 TensorFlow 2.7.0 的新版本的 DLC。 TensorFlow 2.7.0 版本的图像已不再主动维护。有关更多信息，请参阅我的工作负载正在运行时，我的实例会被修补到位吗？有关查找最新 DLC 的步骤，请参阅我怎样才能找到支持的框架版本的最新补丁镜像？

由于补丁升级而创建的新 DLC 将使用更新的发布标签进行指定。如果更改不向后兼容，则标签将更改主要版本而不是次要版本（例如 v1.0 将更改为 v2.0 而不是 v 1.2）。

补丁结束日期（过期）

DLC 的补丁将在 GitHub 发布日期 365 天后到期。

> **Important**
>
> 当有重大框架更新时，我们会例外。例如。如果 TensorFlow 1.15 更新到 TensorFlow 2.0，那么我们将继续支持 TensorFlow 1.15 的最新版本，自 GitHub 发布之日起两年或原始框架维护团队放弃支持后的六个月（以较早的日期为准）。

依赖关系 end-of-support

如果您正在使用 Python 3.6 的 TensorFlow 2.7.0 DLC 映像上运行工作负载，并且该版本的 Python 已标记为，则所有基于 Python 3.6 的 DLC 映像都将不再被主动维护。 end-of-support同样，如果将像 Ubuntu 16.04 这样的操作系统版本标记为 Ubuntu 16.04，则所有依赖于 end-of-support Ubuntu 16.04 的 DLC 镜像都将不再被主动维护。

框架版本已不再主动维护的镜像会被打补丁吗？

否。 不再主动维护的图片将不会发布新版本。

如何使用较旧的框架版本？

要使用框架版本较早的 DLC，请浏览 DLC GitHub 发行标签以找到您选择的图像 URI，然后使用它来提取 docker 镜像。

如何 up-to-date 应对框架及其版本的支持变化？

up-to-date 使用 DLC 发行说明和 “可用的Deep Learning Con tainers 镜像” 页面，继续关注 DL C 框架和版本。

我需要商业许可证才能使用 Anaconda 存储库吗？

Anaconda转向了针对某些用户的商业许可模式。积极维护的 DLC 已从 Anaconda 频道迁移到公开发布的开源版本的 C onda（conda-forge）。

> **Warning**
>
> 如果您正在积极使用 Anaconda 在不再主动维护的 DLC 中安装和管理您的软件包及其依赖关系，则如果您确定这些条款适用于您，则有责任遵守 Anaconda Re pository 的管理许可。或者，您可以迁移到Dee AWSp Learning Containers 框架Support 策略表中列出的当前支持的 DLC 之一，也可以使用 conda-forge 作为源安装软件包。

Deep Learning Containers 映像

AWSDeep Learning Containers 可用作 Amazon ECR 中的 Docker 映像。每个 Docker 映像专用于在带 CPU 或 GPU 支持的特定深度学习框架版本、python 版本上运行训练或推理。

有关可用 Deep Learning Containers 的完整列表以及提取它们的信息，请参阅可用的 Deep Learning Containers 映像。

选择所需的 Deep Learning Containers 图像后，请继续执行以下操作之一：

- 要使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在 Amazon EC2 的 Deep Learning Containers 上运行训练和推理，请参阅Amazon EC2 教程
- 要使用 MXNet、PyTorch 和 TensorFlow 在 Amazon ECS 的 Deep Learning Containers 上运行训练和推理，请参阅Amazon ECS 教
- 适用于 Amazon EKS 的 Deep Learning Containers 提供 CPU、GPU 和基于 GPU 的分布式培训，以及基于 CPU 和 GPU 的推理。要使用 MXNet、PyTorch 和 TensorFlow 在 Amazon EKS 的 Deep Learning Containers 上运行训练和推理，请参阅Amazon EKS
- 有关 Deep Learning Containers 中的安全性的信息，请参阅AWS 深度学习 Containers 中的安全
- 有关最新 Deep Learning Containers er 发布说明的列表，请参阅深度学习容器的发行说明

Deep Learning Containers 资源

以下主题介绍了其他AWSDeep Learning Containers 资源。

目录

构建AWSDeep Learning Containers 自定义映像

如何构建自定义映像

我们可以轻松利用自定义 Deep Learning Containers er，以使用 Docker 文件添加自定义框架、库和程序包。

利用 TensorFlow 进行训练

在以下示例 Dockerfile 中，生成的 Docker 映像将针对 GPU 优化 TensorFlow v1.15.2 并构建用于支持适用于多节点分布式训练的 Horovod 和 Python 3。它也将具有AWS示例 GitHub 存储库，该库包含许多深度学习模型示例。

```
#Take base container
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:1.15.2-gpu-py36-cu100-ubuntu18.04

# Add custom stack of code
RUN git clone https://github.com/aws-samples/deep-learning-models
```

阿帕奇 MXNet 培训（孵化）

在以下示例 Dockerfile 中，生成的 Docker 映像将针对构建用于支持 Horovod 和 Python 3 的 GPU 推理优化 Apache MXNet（孵化）v1.6.0。它还将具有 MXNet GitHub 存储库，该存储库包含许多深度学习模型示例。

```
# Take the base MXNet Container
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-training:1.6.0-gpu-py36-cu101-ubuntu16.04
```

```
# Add Custom stack of code
RUN git clone -b 1.6.0 https://github.com/apache/incubator-mxnet.git

ENTRYPOINT ["python", "/incubator-mxnet/example/image-classification/train_mnist.py"]
```

使用 Docker 映像的自定义名称和自定义标签构建映像，同时指向您的个人 Docker 注册表（通常为您的用户名）。

```
docker build -f Dockerfile -t <registry>/<any name>:<any tag>
```

推送至您的个人 Docker 注册表：

```
docker push <registry>/<any name>:<any tag>
```

您可以使用以下命令运行容器：

```
docker run -it < name or tag>
```

> **Important**
>
> 您可能需要登录才能访问 Deep Learning Containers er 映像存储库。在以下命令中指定您的区域：
>
> ```
> aws ecr get-login-password --region us-east-1 | docker login --username AWS --
> password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com
> ```

AWSDeep Learning Containers 英特尔数学核心库 (MKL) 建议

针对 CPU 容器的 MKL 建议

目录

CPU 实例上深度学习框架的训练和推理工作负载的性能情形不一，具体取决于各种配置设置。例如，在AWSEC2 c5.18xlarge 实例，物理内核数为 36，而逻辑内核数为 72。MKL 的训练和推理的配置设置受这些因素影响。通过更新 MKL 的配置来匹配实例功能，则有可能实现性能改进。

请考虑以下使用 Intel-MKL 优化的 TensorFlow 二进制的示例：

- 据对一个 ResNet50v2 模型（该模型使用 TensorFlow 来训练，训练好后使用TensorFlow Serving 来推理）的观察，当 MKL 设置调整到匹配实例的内核数量时，推理性能提高 2 倍。以下设置用于 c5.18xlarge 实例。

```
export TENSORFLOW_INTER_OP_PARALLELISM=2
# For an EC2 c5.18xlarge instance, number of logical cores = 72
export TENSORFLOW_INTRA_OP_PARALLELISM=72
# For an EC2 c5.18xlarge instance, number of physical cores = 36
export OMP_NUM_THREADS=36
export KMP_AFFINITY='granularity=fine,verbose,compact,1,0'
# For an EC2 c5.18xlarge instance, number of physical cores / 4 = 36 /4 = 9
export TENSORFLOW_SESSION_PARALLELISM=9
export KMP_BLOCKTIME=1
export KMP_SETTINGS=0
```

- 据对一个 ResNet50_v1.5 模型（该模型使用 TensorFlow 在 ImageNet 数据集上训练，并且使用 NHWC 图像形状）的观察，训练吞吐量性能加快了 9 倍左右。比较对象为未进行 MKL 优化的二级制，用“样本数/秒”指标来进行度量。使用以下环境变量：

```
export TENSORFLOW_INTER_OP_PARALLELISM=0
# For an EC2 c5.18xlarge instance, number of logical cores = 72
export TENSORFLOW_INTRA_OP_PARALLELISM=72
# For an EC2 c5.18xlarge instance, number of physical cores = 36
export OMP_NUM_THREADS=36
export KMP_AFFINITY='granularity=fine,verbose,compact,1,0'
# For an EC2 c5.18xlarge instance, number of physical cores / 4 = 36 /4 = 9
export KMP_BLOCKTIME=1
export KMP_SETTINGS=0
```

以下链接将帮助您了解如何调整 Intel MKL 和您的深度学习框架设置，以优化深度学习工作负载：

- Intel 优化的 TensorFlow Serving 的一般最佳实践
- TensorFlow 性能
- 提高 Apache MXNet 性能的一些技巧
- 包含 Intel MKL-DNN 的 MXNet 的性能基准测试

设置环境变量的 EC2 指南

请参阅 `docker run` 文档，了解在创建容器时如何设置环境变量：https://docs.docker.com/engine/reference/run/#env-environment-variables

下面是为 docker run 设置称为 `OMP_NUM_THREADS` 的环境变量的一个示例。

```
ubuntu@ip-172-31-95-248:~$ docker run -e OMP_NUM_THREADS=36 -it --entrypoint ""
 999999999999.dkr.ecr.us-east-1.amazonaws.com/beta-tensorflow-inference:1.13-py2-cpu-
build bash
root@d437faf9b684:/# echo $OMP_NUM_THREADS
36
```

在极少数情况下，英特尔 MKL 可能会产生不良影响。若要禁用 MKL TensorFlow 请设置以下环境变量：

```
export TF_DISABLE_MKL=1
export TF_DISABLE_POOL_ALLOCATOR=1
```

设置环境变量的 ECS 指南

要在 ECS 中为容器指定运行时环境变量，必须编辑 ECS 任务定义。在任务定义的 `containerDefinitions` 部分，以“名称-值”密钥对的形式添加环境变量。下面为设置 `OMP_NUM_THREADS` 和 `KMP_BLOCKTIME` 变量的示例。

```
{
```

```
    "requiresCompatibilities": [
        "EC2"
    ],
    "containerDefinitions": [{
        "command": [
            "mkdir -p /test && cd /test && git clone -b r1.13 https://github.com/
tensorflow/serving.git && tensorflow_model_server --port=8500 --rest_api_port=8501
 --model_name=saved_model_half_plus_two_cpu --model_base_path=/test/serving/
tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_cpu"
        ],
        "entryPoint": [
            "sh",
            "-c"
        ],
        "name": "EC2TFInference",
        "image": "999999999999.dkr.ecr.us-east-1.amazonaws.com/tf-inference:1.12-cpu-
py3-ubuntu16.04",
        "memory": 8111,
        "cpu": 256,
        "essential": true,
        "environment": [{
              "name": "OMP_NUM_THREADS",
              "value": "36"
            },
            {
              "name": "KMP_BLOCKTIME",
              "value": 1
            }
        ],
        "portMappings": [{
                "hostPort": 8500,
                "protocol": "tcp",
                "containerPort": 8500
            },
            {
                "hostPort": 8501,
                "protocol": "tcp",
                "containerPort": 8501
            },
            {
                "containerPort": 80,
                "protocol": "tcp"
            }
        ],
```

```
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/TFInference",
                "awslogs-region": "us-west-2",
                "awslogs-stream-prefix": "ecs",
                "awslogs-create-group": "true"
            }
        }
    }],
    "volumes": [],
    "networkMode": "bridge",
    "placementConstraints": [],
    "family": "Ec2TFInference"
}
```

在极少数情况下，英特尔 MKL 可能会产生不良影响。若要禁用 MKL TensorFlow 请设置以下环境变量：

```
            {
              "name": "TF_DISABLE_MKL",
              "value": 1
            },
            {
              "name": "TF_DISABLE_POOL_ALLOCATOR",
              "value": 1
            }
```

设置环境变量的 EKS 指南

要为容器指定运行时环境变量，编辑 EKS 作业（.yaml、.json）的原始清单。以下清单的代码段显示名为 `squeezenet-service` 的容器的定义。除了其他属性如 args 和端口外，环境变量以“名称-值”密钥对的形式列出。

```
      containers:
      - name: squeezenet-service
        image: 999999999999.dkr.ecr.us-east-1.amazonaws.com/beta-mxnet-inference:1.4.0-
py3-gpu-build
        command:
        - mxnet-model-server
```

```
        args:
        - --start
        - --mms-config /home/model-server/config.properties
        - --models squeezenet=https://s3.amazonaws.com/model-server/models/
squeezenet_v1.1/squeezenet_v1.1.model
        ports:
        - name: mms
          containerPort: 8080
        - name: mms-management
          containerPort: 8081
        imagePullPolicy: IfNotPresent
        env:
        - name: AWS_REGION
          value: us-east-1
        - name: OMP_NUM_THREADS
          value: 36
        - name: TENSORFLOW_INTER_OP_PARALLELISM
          value: 0
        - name: KMP_AFFINITY
          value: 'granularity=fine,verbose,compact,1,0'
        - name: KMP_BLOCKTIME
          value: 1
```

在极少数情况下，英特尔 MKL 可能会产生不良影响。若要禁用 MKL TensorFlow 请设置以下环境变量：

```
- name: TF_DISABLE_MKL
          value: 1
- name: TF_DISABLE_POOL_ALLOCATOR
          value: 1
```

AWS 深度学习 Containers 中的安全

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于 Deep Learning Containers 的合规计划，请参阅按合规计划划分的范围内的AWSAWS 服务服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用 Deep Learning Containers 时如何应用分担责任模型。以下主题向您展示了如何配置 Deep Learning Containers 以实现您的安全和合规目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Deep Learning Containers 资源。

有关更多信息，请参阅 Amazon EC2 中的安全、Amazon ECS 中的安全、Amazon EK S 中的安全以及亚马逊的安全 SageMaker。

主题

- AWS 深度学习 Containers 中的数据保护
- AWS 深度学习容器中的身份和访问管理 Deep Learning Containers
- Dee AWS p Learning Containers 中的监控和使用跟踪
- AWS 深度学习 Containers 的合规性验证
- AWS 深度学习 Containers 中的弹性
- AWS 深度学习 Containers 中的基础设施安全

AWS 深度学习 Containers 中的数据保护

分担责任模型 AWS 分担责任模型适用于Dee AWS p Learning Containers中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请

参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括你使用控制台、API 或 SDK AWS 服务 使用 AWS Deep Learning Contain AWS CLI ers 或其他容器时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

AWS 深度学习容器中的身份和访问管理 Deep Learning Containers

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用深度学习容器资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

有关身份和访问管理的更多信息，请参阅 Amazon EC2 的身份和访问管理、Amazon EC S 的身份和访问管理、亚马逊 EKS 的身份和访问管理，以及亚马逊的身份和访问管理，以及亚马逊的身份和访问管理 SageMaker。

主题

- 使用身份进行身份验证
- 使用策略管理访问

- IAM 与 Amazon EMR 结合使用

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南 中的 需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

- 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括

AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》 中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 - 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

IAM 与 Amazon EMR 结合使用

您可以 AWS Identity and Access Management 与 Amazon EMR 一起使用来定义用户、 AWS 资源、群组、角色和策略。您还可以控制这些用户和角色可以访问哪些 AWS 服务。

有关将 IAM 与 Amazon EMR 结合使用的更多信息，请参阅 Amazon EMR 的AWS Identity and Access Management。

Dee AWS p Learning Containers 中的监控和使用跟踪

你的 D AWS eep Learning Containers 不附带监控工具。有关监控的信息，请参阅 GPU 监控和优化、监控 Amazon EC2、监控 Amazon EC S、监控 Ama zon EKS 和监控 Amazon SageMaker Studio。

使用情况跟踪

AWS 使用客户反馈和使用信息来提高我们向客户提供的服务和软件的质量。我们在支持的 Dee AWS p Learning Containers 中增加了使用数据收集功能，以便更好地了解客户的使用情况并指导未来的改进。默认情况下，Deep Learning Containers 的使用情况跟踪处于激活状态。客户可以随时更改其设置，以激活或停用使用情况跟踪。

Dee AWS p Learning Containers 的使用情况跟踪会收集用于容器的实例 ID、框架、框架版本、容器类型和 Python 版本。 AWS 还会记录它接收此元数据的事件时间。

不会收集或保留有关容器内使用的命令的信息。不会收集或保留有关容器的其他信息。

要选择退出使用情况跟踪，请将`OPT_OUT_TRACKING`环境变量设置为 true。

```
OPT_OUT_TRACKING=true
```

故障率跟踪

使用第一方 Dee Amazon SageMaker AWS p Learning Containers 容器时， SageMaker 团队将收集故障率元数据以提高 AWS 深度学习容器的质量。默认情况下，Dee AWS p Learning Containers 的故障率跟踪处于活动状态。客户可以在创建 Amazon SageMaker 端点时更改其设置以激活或停用故障率跟踪。

Dee AWS p Learning Containers 的故障率跟踪会收集实例 ID、ModelServer 名称ErrorType、ModelServer 版本和ErrorCode。 AWS 还会记录它接收此元数据的事件时间。

不会收集或保留有关容器内使用的命令的信息。不会收集或保留有关容器的其他信息。

要选择退出故障率跟踪，请将OPT_OUT_TRACKING环境变量设置为true。

```
OPT_OUT_TRACKING=true
```

以下框架版本中的使用情况跟踪

不再支持以下框架版本：

- TensorFlow 1.15
- TensorFlow 2.0
- TensorFlow 2.1
- PyTorch 1.2
- PyTorch 1.3.1
- MxNet 1.6

有关我们支持政策的完整说明，请参阅框架Support 政策。

虽然我们建议更新到支持的 Deep Learning Containers，但要选择退出使用这些框架的 Deep Learning Containers 的使用情况跟踪，请将OPT_OUT_TRACKING环境变量设置为 true，然后使用自定义入口点来禁用对以下服务的调用：

- 亚马逊 EC2 自定义入口点
- Amazon ECS 自定义入口点
- 亚马逊 EKS 自定义入口点

AWS 深度学习 Containers 的合规性验证

作为多个合规计划的一部分，第三方审计师对服务的安全性和 AWS 合规性进行评估。有关支持的合规计划的信息，请参阅 Amazon EC2 的合规性验证、Amazon EC S 的合规性验证、A mazon E KS 的合规性验证和亚马逊的合规验证 SageMaker。

有关特定合规计划范围内的 AWS 服务列表，请参阅按合规计划划分的范围内的AWSAWS 服务按合规计划。有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅在 Artifact 中下载报告在 AWS Ar 。

您在使用 Deep Learning Containers 时的合规责任取决于您的数据的敏感度、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全性与合规性快速入门指南 - 这些部署指南讨论了架构注意事项，并提供了在 AWS上部署基于安全性和合规性的基准环境的步骤。
- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业指导方针和法规。
- AWS Security Hub— 此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

AWS 深度学习 Containers 中的弹性

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

有关有助于支持您的数据弹性和备份需求的功能的信息，请参阅 Amazon EC2 中的弹性、Amaz on EKS 中的弹性和亚马逊的弹性。 SageMaker

AWS 深度学习 Containers 中的基础设施安全

Dee AWS p Learning Containers 的基础设施安全由亚马逊 EC2、Amazon ECS、Amazon EKS 或提供支持 SageMaker。有关更多信息，请参阅 Amazon EC2 中的基础设施安全、Amazon ECS 中的基础设施安全、Amazon E KS 中的基础设施安全和亚马逊的基础设施安全 SageMaker。

深度学习容器的发行说明

查看为特定机器学习框架、基础架构和 AWS 服务构建的 Dee AWS p Learning Containers 的最新版本说明。

> **Note**
>
> 从 PyTorch 1.10 和 Tensorflow 2.7 开始，CPU 和 GPU Deep Learning Containers 以 EC2、ECS 和 EKS 镜像的形式发布。SageMaker

单框架深度学习容器 Deep Learning

TensorFlow

- AWS 适用于 TensorFlow 2.13 的 Deep Learning Containers（培训开启 SageMaker）：2023 年 8 月 7 日
- AWS 适用于 TensorFlow 2.13 的 Deep Learning Containers（推断开启 SageMaker）：2023 年 8 月 8 日
- AWS 适用于 TensorFlow 2.13 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2023 年 7 月 19 日
- AWS 适用于 TensorFlow 2.13 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2023 年 8 月 3 日
- AWS 适用于 TensorFlow 2.12 的 Deep Learning Containers（推断开启 SageMaker）：2023 年 5 月 18 日
- AWS 适用于 TensorFlow 2.12 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2023 年 5 月 13 日
- AWS 适用于 TensorFlow 2.12 的 Deep Learning Containers（培训开启 SageMaker）：2023 年 4 月 11 日
- AWS 适用于 TensorFlow 2.12 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行训练）：2023 年 3 月 23 日
- AWS 适用于 TensorFlow 2.11 的 Deep Learning Containers（培训开启 SageMaker）：2023 年 1 月 10 日
- AWS 适用于 TensorFlow 2.11 的 Deep Learning Containers（推理开启 SageMaker）：2022 年 12 月 5 日

- AWS 适用于 TensorFlow 2.11 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2022 年 12 月 5 日
- AWS 适用于 TensorFlow 2.11 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2022 年 12 月 5 日
- AWS 适用于 TensorFlow 2.10 的 Deep Learning Containers（推理开启 SageMaker）：2022 年 9 月 30 日
- AWS 适用于 TensorFlow 2.10 的 Deep Learning Containers（培训开启 SageMaker）：2022 年 9 月 30 日
- AWS 适用于 TensorFlow 2.10 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2022 年 9 月 15 日
- AWS 适用于 TensorFlow 2.10 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2022 年 5 月 19 日
- AWS TensorFlow 2.9 版 Deep Learning Containers（推理开启 SageMaker）：2022 年 6 月 18 日
- AWS TensorFlow 2.9 版 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2022 年 6 月 18 日
- AWS TensorFlow 2.9 版 Deep Learning Containers（培训开启 SageMaker）：2022 年 6 月 13 日
- AWS 适用于 TensorFlow 2.9 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2022 年 5 月 19 日
- AWS TensorFlow 2.8 版 Deep Learning Containers (SageMaker)：2022 年 3 月 22 日
- AWS 适用于 TensorFlow 2.8 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 3 月 22 日
- AWS TensorFlow 2.7 版 Deep Learning Containers (SageMaker)：2022 年 3 月 22 日
- AWS 适用于 TensorFlow 2.7 的 Deep Learning Containers（EC2、ECS 和 EKS）：2021 年 12 月 15 日
- AWS TensorFlow 2.6 版 Deep Learning Containers：2021 年 9 月 24
- AWS TensorFlow 2.5 版深度学习容器：2021 年 7 月 1 日
- AWS TensorFlow 2.4 版 Deep Learning Containers：2021 年 3 月 15
- AWS TensorFlow 2.3 版 Deep Learning Containers：2021 年 3 月 15
- AWS 带 CUDA 11.0 的 TensorFlow 2.3 版 Deep Learning Containers：2020 年 10 月 15 日
- AWS TensorFlow 2.3 版 Deep Learning Containers：2020 年 8 月 7
- AWS TensorFlow 2.2 版 Deep Learning Containers：2021 年 3 月 15
- AWS 带 CUDA 10. TensorFlow 2 的 2.2 版 Deep Learning Containers：2020 年 7 月 24 日

- AWS 带 CUDA 10.1 的 TensorFlow 2.2 版 Deep Learning Containers：2020 年 7 月 20 日
- AWS 带 CUDA 10.1 的 TensorFlow 2.2 版 Deep Learning Containers：2020 年 5 月 20 日
- AWS TensorFlow 2.1 版 Deep Learning Containers：2021 年 3 月 15
- AWS TensorFlow 2.1 版 Deep Learning Containers：2020 年 6 月 25
- AWS 适用于 Tensorflow 的深度学习容器 2.1：2020 年 3 月 19 日
- AWS 适用于 TensorFlow 2.0 的深度学习容器：2021 年 3 月 15 日
- AWS 适用于 TensorFlow 2.0 的深度学习容器：2020 年 6 月 19 日
- AWS TensorFlow 2.0 版 Deep Learning Containers：2020 年 2 月 26
- AWS 适用于 Tensorflow 的深度学习容器 1.15 2020 年 7 月 29 日
- AWS 支持 python-3.7 的 TensorFlow 1.15 版 Deep Learning Containers：2020 年 5 月 6 日
- AWS 适用于 Tensorflow 的 Deep Learning 容器 1.15 2020 年 3 月 19 日

PyTorch

- AWS 适用于 PyTorch 2.3 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2024 年 5 月 30 日
- AWS 适用于 PyTorch 2.3 的 Deep Learning Containers（正在训练 SageMaker）：2024 年 5 月 30 日
- AWS 适用于 PyTorch 2.2 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2024 年 3 月 20 日
- AWS PyTorch 2.2 版 Deep Learning Containers（培训开启 SageMaker）：2024 年 3 月 20 日
- AWS PyTorch 2.2 版 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2024 年 3 月 20 日
- AWS PyTorch 2.2 版 Deep Learning Containers（推理开启 SageMaker）：2024 年 3 月 20 日
- AWS 适用于 PyTorch 2.1 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行训练）：2023 年 11 月 7 日
- AWS PyTorch 2.1 版 Deep Learning Containers（培训开启 SageMaker）：2023 年 11 月 7 日
- AWS 适用于 PyTorch 2.1 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2023 年 11 月 7 日
- AWS PyTorch 2.1 版 Deep Learning Containers（推理开启 SageMaker）：2023 年 11 月 7 日
- AWS 带有 CUDA 12.1 的 PyTorch 2.0.1 版 Deep Learning Containers（培训开启 SageMaker）：2023 年 11 月 2 日

- AWS 带有 CUDA 12.1 的 PyTorch 2.0.1 版 Deep Learning Containers（在 EC2、ECS 和 EKS 上训练）：2023 年 9 月 6 日
- AWS 适用于 PyTorch 2.0 的 Deep Learning Containers（培训开启 SageMaker）：2023 年 4 月 18 日
- AWS 适用于 PyTorch 2.0 的 Deep Learning Containers（推理开启 SageMaker）：2023 年 4 月 11 日
- AWS 适用于 PyTorch 2.0 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行训练）：2023 年 3 月 30 日
- AWS 适用于 PyTorch 2.0 的 Deep Learning Containers（在 EC2、ECS 和 EKS 上进行推理）：2023 年 3 月 29 日
- AWS 适用于 PyTorch 1.13.1 的 Deep Learning Containers (SageMaker)：2023 年 1 月 23 日
- AWS 适用于 PyTorch 1.13 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 11 月 9 日
- AWS 适用于 PyTorch 1.12 的 Deep Learning Contain SageMaker ers ()：2022 年 12 月 15 日
- AWS 适用于 PyTorch 1.12 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 12 月 15 日
- AWS 适用于 PyTorch 1.11 的 Deep Learning Contain SageMaker ers ()：2022 年 5 月 6 日
- AWS 适用于 PyTorch 1.11 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 4 月 14 日
- AWS 适用于 PyTorch 1.10 的 Deep Learning Contain SageMaker ers ()：2022 年 4 月 14 日
- AWS 适用于 PyTorch 1.10 的 Deep Learning Containers（EC2、ECS 和 EKS）：2021 年 11 月 3 日
- AWS 适用于 PyTorch 1.9 的深度学习容器：2021 年 8 月 30 日
- AWS PyTorch 1.8 版 Deep Learning Containers：2021 年 3 月 16
- AWS 带 CUDA 11.0 的 PyTorch 1.7 版 Deep Learning Containers：2021 年 3 月 15 日
- AWS 带 CUDA 11.0 的 PyTorch 1.6 版 Deep Learning Containers：2020 年 12 月 9 日
- AWS 适用于 PyTorch 1.6 的深度学习容器：2020 年 8 月 3 日
- AWS PyTorch 1.5 版 Deep Learning Containers：2020 年 6 月 19
- AWS PyTorch 1.5 版 Deep Learning Containers：2020 年 5 月 5
- AWS 适用于 PyTorch 1.4 的深度学习容器：2020 年 6 月 6 日
- AWS 适用于 PyTorch 1.4 的深度学习容器：2020 年 4 月 3 日

Graviton 深度学习容器

TensorFlow

- AWS 适用于 Graviton 的 Deep Learning Conta TensorFlow iners 2.13 (SageMaker)：2023 年 8 月 17 日
- AWS 适用于 Graviton TensorFlow 2.13 的 Deep Learning Containers（EC2、ECS 和 EKS）：2023 年 8 月 17 日
- AWS 适用于 Graviton TensorFlow 2.12 的 Deep Learning Containers (SageMaker)：2023 年 5 月 18 日
- AWS 适用于 Graviton TensorFlow 2.12 的 Deep Learning Containers（EC2、ECS 和 EKS）：2023 年 5 月 18 日
- AWS 适用于 Graviton 的 Deep Learning Conta TensorFlow iners 2.9 (SageMaker)：2022 年 10 月 20 日
- AWS 适用于 Graviton TensorFlow 2.9 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 8 月 29 日
- AWS 适用于 Graviton TensorFlow 2.7 的 Deep Learning Containers（EC2、ECS 和 EKS）：2021 年 12 月 4 日

PyTorch

- AWS 适用于 Graviton 的 Deep Learning Conta PyTorch iners 2.2 (SageMaker)：2024 年 4 月 16 日
- AWS 适用于 Graviton PyTorch 2.2 的 Deep Learning Containers（EC2、ECS 和 EKS）：2024 年 4 月 16 日
- AWS 适用于 Graviton 的 Deep Learning Conta PyTorch iners 2.1 (SageMaker)：2023 年 10 月 25 日
- AWS 适用于 Graviton PyTorch 2.1 的 Deep Learning Containers（EC2、ECS 和 EKS）：2023 年 10 月 25 日
- AWS 适用于 Graviton PyTorch 2.0 的 Deep Learning Containers (SageMaker)：2023 年 4 月 11 日
- AWS 适用于 Graviton PyTorch 2.0 的 Deep Learning Containers（EC2、ECS 和 EKS）：2023 年 3 月 29 日
- AWS 适用于 Graviton PyTorch 1.12 的 Deep Learning Containers (SageMaker)：2022 年 10 月 20 日

- AWS 适用于 Graviton PyTorch 1.12 的 Deep Learning Containers（EC2、ECS 和 EKS）：2022 年 8 月 29 日
- AWS 适用于 Graviton PyTorch 1.10 的 Deep Learning Containers（EC2、ECS 和 EKS）：2021 年 12 月 4 日

Deep Learning Containers 文档历史记录开发者指南

下表介绍了此版本的Deep Learning Containers 文档。

- API 版本：最新
- 文档最新更新时间：2020 年 2 月 26 日

变更	说明	日期
Apache MXNet 和 Horovod	Apache MXNet 教程已添加到开发者指南中。	2020 年 2 月 26 日
《Deep Learning Containers 开发指南》发布会	Deep Learning Containers 设置和教程已添加到开发者指南中。	2020 年 2 月 17 日

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。