AmazonDeep Learning 容器

开发人员指南

AmazonDeep Learning 容器: 开发人员指南

Table of Contents

什么是AmazonDeep Learning Containers？

欢迎阅读用户指南AmazonDeep Learning Containers。

AmazonDeep Learning 容器（Deep Learning Containers）是 TensorFlow、TensorFlow 2、PyTorch 和 Apache MXNet（孵化）中训练和处理模型的一组 Docker 映像。Deep Learning Containers 提供具有 TensorFlow 和 MXNet、Nvidia CUDA（适用于 GPU 实例）和 Intel MKL（适用于 CPU 实例）库的优化环境，并在 Amazon Elastic Container Registry (Amazon ECR) 中可用。

AmazonDeep Learning Containers | Amazon Web Services

关于本指南

本指南可以帮助您设置和使用AmazonDeep Learning Containers。本指南还介绍了使用 Amazon EC2、Amazon ECS、Amazon EKS 和 SageMaker 设置 Deep Learning Containers。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。本指南还针对每个框架提供多个教程。

- 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 (p. 2)
- 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 教程 (p. 12)
- 适用于 Amazon EKS 的 Deep Learning Containers 提供基于 CPU、GPU 和分布式 GPU 的培训，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 教程 (p. 34)
- 有关基于 Docker 的 Deep Learning Containers 映像的说明、可用映像列表以及如何使用它们，请参阅Deep Learning Containers 映像 (p. 85)
- 有关 Deep Learning Containers 的安全性的信息，请参阅中的安全性AmazonDeep Learning 容器 (p. 91)
- 有关最新 Deep Learning Containers 发行说明的列表，请参阅Deeeep Learning 容器 (p. 97)

Python 2 Support

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 版本将是支持 Python 2 的最后版本。支持 Python 2 的先前版本的 Deep Learning Containers 将继续可用。但是，只有在开源社区针对 Python 2 Deep Learning Containers 发布了安全修补程序时，我们才会为这些版本提供更新。随着 TensorFlow 和 PyTorch 框架的下一个版本发布的 Deep Learning Containers 将不包括 Python 2 环境。

先决条件

您应该熟悉命令行工具和基本 Python 才能成功运行 Deep Learning Containers。有关如何使用每个框架的教程由框架本身提供。但是，本指南向您展示如何激活每个教程并找到适当的入门教程。

Deep Learning Containers 入门

以下部分介绍了如何使用 Deep Learning Containers 从Amazon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

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 (p. 2).

目录

Amazon EC2 设置

在本节中，您将了解如何设置Amazon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 Amazon 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Amazon CLI使用中的步骤保持最新状态安装最新的AmazonCLI 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Amazon请参阅 Deep Learning ContainersDeep Learning Containers 映像 (p. 85).

Note

MKL 用户：读取AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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 (p. 2).

Training

本部分演示如何在上运行训练Amazon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 (p. 85).

Note

MKL 用户：读取AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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 (p. 7).

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） (p. 9).

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：

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 (p. 10).

推理

此节演示如何在上运行推理Amazon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AmazonDeep Learning Containers。有关 Elastic Inference 的教程和更多信息，请参阅使用 AmazonAmazon 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 (p. 85).

> **Note**
>
> MKL 用户：读取 AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)以获得最佳训练或推理性能。

目录

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 (p. 34).

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 教程

本部分演示如何在上运行训练和推理。Amazon使用 MXNet、PyTorch 和 TensorFlow 的 Amazon ECS 的 Deep Learning Containers。

在开始以下教程之前，请完成中的步骤。Amazon ECS 设置 (p. 12).

要获取 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 85).

> Note
>
> MKL 用户：读取AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)获得最佳训练或推理性能。

目录

Amazon ECS 设置

本主题介绍如何设置AmazonDeep Learning Containers 使用 Amazon Elastic Container Service。

目录

先决条件

本安装指南假设您已完成以下先决条件：

- 安装并配置最新版本的Amazon CLI. 有关安装或升级 Amazon CLI 的更多信息，请参阅安装 Amazon Command Line Interface。
- 完成中的步骤使用 Amazon ECS 进行设置.
- 以下情况之一是真的：
  - 您的用户拥有管理员权限。有关更多信息，请参阅使用 Amazon ECS 进行设置。
  - 您的用户拥有创建服务角色的 IAM 权限。有关更多信息，请参阅创建角色以向 Amazon 服务委派权限。
  - 拥有管理员权限的用户手动创建了这些 IAM 角色，使之对所用的账户可用。有关更多信息，请参阅 。Amazon ECS 服务计划程序 IAM 角色和Amazon ECS 容器实例 IAM 角色中的Amazon Elastic Container Service 开发人员.
- Amazon CloudWatch 将日志 IAM 策略添加到 Amazon ECS 容器实例 IAM 角色，从而允许 Amazon ECS 将日志发送到 Amazon CloudWatch。有关更多信息，请参阅 。CloudWatch Logs IAM 策略中的Amazon Elastic Container Service 开发人员.
- 生成密钥对。有关更多信息，请参阅Amazon EC2 密钥对.
- 创建一个新的安全组或更新现有安全组，使之对所需的推理打开端口。
  - 对于 MXNet 推理，对于 TCP 流量打开端口 80 和 8081。
  - 适用于 TensorFlow 推理，对于 TCP 流量打开端口 8501 和 8500。

  有关更多信息，请参阅Amazon EC2 安全组.

为 Deep Learning Containers 设置 Amazon ECS

本部分介绍如何将 Amazon ECS 设置为使用 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网络模式或者服务配置为使用以下任何一种方式：服务发现、外部部署控制器、多个目标组或 Elastic Inference 加速器。如果是这种情况，则不应在此处指定一个角色。有关更多信息，请参阅 。对 Amazon ECS 使用服务相关角色中的Amazon ECS 开发人员指南.

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 Amazon ECS 集群。

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

2. 在集群中启动一个或多个 Amazon EC2 实例。有关基于 GPU 的工作，请参阅在 Amazon ECS 上使用 GPU中的Amazon ECS 开发人员指南以通知您的实例类型选择。如果您选择 GPU 实例类型，请确保选择经 Amazon ECS 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经 Amazon ECS 优化的 AMI ID 的更多信息，请参阅经 Amazon ECS 优化的 AMI. 在本示例中，您将在 us-east-1 中启动一个具有 100 GB 磁盘大小的 AMI 的实例。

   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. 使用经 Amazon ECS 优化的 AMI 启动 Amazon EC2 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，请参阅经 Amazon ECS 优化的 AMI中的Amazon Elastic Container Service 开发人员.

```
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，您可以使用`instance-id`从响应中显示。

现在，您有一个正在运行容器实例的 Amazon ECS 集群。按照以下步骤验证 Amazon EC2 实例是否已注册到集群。

验证 Amazon EC2 实例是否已注册到集群

1. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 控制台。
2. 选择具有注册 Amazon EC2 实例的集群。
3. 在 Cluster 页面上，选择 ECS Instances。
4. 验证客服已连接值是True(对于)`instance-id`在上一步中创建。此外，请注意控制台上显示的 CPU 可用和内存值因为在随后的教程中会用到这些值。上述值可能需要几分钟才能显示在控制台中。

后续步骤

要了解如何在 Amazon ECS 上使用 Deep Learning Containers 进行训练和推理，请参阅Amazon ECS 教程 (p. 12).

培训

本部分介绍如何在上运行训练。Amazon适用于 Amazon Elastic Containers 服务的 Deep Learning Containers 使用 Apache MXNet、TensorFlow 和 TensorFlow 2。

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 85).

> Note
>
> MKL 用户：读取AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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。如果您的任务定义使用awsvpc网络模式，或者将服务配置为使用服务发现。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，您不

应在此处指定角色），则需要该角色。有关更多信息，请参阅 。Amazon ECS 使用服务相关角色中的Amazon ECS 开发人员指南.

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。以下示例使用将训练脚本添加到 Deep Learning Containers 的示例 Docker 映像。您可以将此脚本与 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。

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

   - 对于 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。记下输出中的修订号，然后在下一个步骤中使用它。

```
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以及在安装过程中创建的集群的名称

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

4. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 在你的任务进入RUNNING状态中，选择任务标识符。
8. 在 Containers (容器) 下，展开容器详细信息。

9. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

后续步骤

要了解在 Amazon ECS 上使用带有 Deep Learning Containers 的 TensorFlow 进行推理，请参阅TensorFlow 推理 (p. 24).

Apache MXNet (孵化版) 训练

您必须先注册任务定义，然后才能在 Amazon Elastic Containers Services 集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用将训练脚本添加到 Deep Learning Containers 的示例 Docker 映像。

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

   - 对于 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。记下输出中的修订号，然后在下一个步骤中使用它。

```
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。您需要使用上一步中的修订号。

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

4. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 在你的任务进入RUNNING状态中，选择任务标识符。
8. 在 Containers (容器) 下，展开容器详细信息。
9. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

后续步骤

要了解使用带有 Deep Learning Containers 的 MxNet 在 Amazon ECS 上的推理，请参阅Apache MXNet (孵化) 推理 (p. 27).

PyTorch 训练

您必须先注册任务定义，然后才能在 Amazon ECS 集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用将训练脚本添加到 Deep Learning Containers 的示例 Docker 映像。

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

   - 对于 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。记下输出中的修订号，然后在下一个步骤中使用它。

```
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。您需要使用上一步中的修订标识符。

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

4. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 在你的任务进入RUNNING状态中，选择任务标识符。
8. 在 Containers (容器) 下，展开容器详细信息。
9. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

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 Amazon ECS 中使用 Amazon S3 插件，请设置AWS_REGION环境变量与您选择的区域。

```
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。

   - 对于 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。记下输出中的修订号，然后在下一个步骤中使用它。

```
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。您需要使用上一步中的修订标识符。

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

5. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 控制台。
6. 选择 `ecs-pytorch-s3-plugin` 集群。
7. 在 Cluster 页面上，选择 Tasks。
8. 在你的任务进入`RUNNING`状态中，选择任务标识符。
9. 在 Containers (容器) 下，展开容器详细信息。
10. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看 Amazon S3 插件示例日志。

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

后续步骤

要了解在 Amazon ECS 上使用 PyTorch 和 Deep Learning Containers 的推理，请参阅PyTorch 推理 (p. 31).

Inference

本节演示如何在上运行推理Amazon使用 Apache MXNet（孵化）、PyTorch、TensorFlow 和 TensorFlow 2 的 Deep Learning Containers。你也可以使用 Elastic Inference 来运行推理AmazonDeep Learning Containers。有关 Elastic Inference 的教程和更多信息，请参阅使用AmazonAmazon ECS 上的 Elastic Inference Deep Learning Containers.

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 85).

> Note
>
> MKL 用户：读取AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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。如果您的任务定义使用awsvpc网络模式。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，您不应在此处指定角色），则需要该角色。有关更多信息，请参阅 。对 Amazon ECS 使用服务相关角色中的Amazon ECS 开发人员指南.

目录

TensorFlow 推理

以下示例使用将 CPU 或 GPU 推理脚本从主机的命令行添加到 Deep Learning Containers 的示例 Docker 映像。

基于 CPU 的推理

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。您可以将它与 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，并克隆 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。记下输出中的修订号并在下一步中使用该修订号。

```
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，请替换`revision_id`其中包含上一步的输出中任务定义的修订号。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 晚于你的任务在 `RUNNING`州，选择任务标识符.

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在名称然后网络绑定，在外部链接注意端口 8501 的 IP 地址并且在下一个步骤中使用它。

   g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

5. 要运行推理，请使用以下命令。将外部 IP 地址替换为上一步中的外部链接 IP 地址。

```
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，请确保您的企业防火墙不会阻止非标准端口，如 8501。您可以尝试切换至来宾网络来验证。

基于 GPU 的推理

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。您可以将它与 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，并克隆 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。记下输出中的修订号并在下一步中使用该修订号。

```
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，请替换`revision_id`其中包含上一步的输出中任务定义的修订号。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 在你的任务进入RUNNING状态中，选择任务标识符。
   e. 在 Containers (容器) 下，展开容器详细信息。
   f. 在名称然后网络绑定，在外部链接记下端口 8501 的 IP 地址并在下一步中使用该 IP 地址。
   g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

5. 要运行推理，请使用以下命令。将外部 IP 地址替换为上一步中的外部链接 IP 地址。

```
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，请确保您的企业防火墙不会阻止非标准端口，如 8501。您可以尝试切换至来宾网络来验证。

Apache MXNet (孵化) 推理

您必须先注册任务定义，然后才能在 Amazon ECS 集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用将 CPU 或 GPU 推理脚本从主机的命令行添加到 Deep Learning Containers 的示例 Docker 映像。

基于 CPU 的推理

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。记下输出中的修订号并在下一步中使用该修订号。

```
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，请替换`revision_id`其中包含上一步的输出中任务定义的修订号。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 在你的任务进入`RUNNING`状态中，选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在名称然后网络绑定，在外部链接记下端口 8081 的 IP 地址并在下一步中使用该 IP 地址。

   g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

5. 要运行推理，请使用以下命令。替换`external IP`上一步中的外部链接 IP 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 的推理

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。记下修订号的输出并在下一步中使用它。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 在你的任务进入`RUNNING`状态中，选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在名称然后网络绑定，在外部链接记下端口 8081 的 IP 地址并在下一步中使用该 IP 地址。

   g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

4. 要运行推理，请使用以下命令。替换`external IP`上一步中的外部链接 IP 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 推理

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

基于 CPU 的推理

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。记下输出中的修订号并在下一步中使用该修订号。

```
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，请替换`revision_id`其中包含上一步的输出中任务定义的修订号。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 在你的任务进入`RUNNING`状态中，选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在名称然后网络绑定，在外部链接记下端口 8081 的 IP 地址并在下一步中使用该 IP 地址。

   g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

5. 要运行推理，请使用以下命令。替换`external IP`上一步中的外部链接 IP 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 的推理

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。记下修订号的输出并在下一步中使用它。

```
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. 打开 https://console.aws.amazon.com/ecs/ 上的 Amazon ECS 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. 一旦你的任务进入RUNNING状态中，选择任务标识符。

e. 在 Containers (容器) 下，展开容器详细信息。

f. 在名称然后网络绑定，在外部链接记下端口 8081 的 IP 地址并在下一步中使用该 IP 地址。

g. 在 Log Configuration (日志配置) 下，选择 View logs in CloudWatch (查看 CloudWatch 中的日志)。这会将您转到 CloudWatch 控制台以查看训练进度日志。

4. 要运行推理，请使用以下命令。替换external IP上一步中的外部链接 IP 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 结合使用，请参阅自定义入口点 (p. 34).

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 教程

本部分演示如何在上运行训练和推理AmazonEKS 的 Deep Learning Containers 使用 MXNet、PyTorch 和 TensorFlow。有些示例包含单节点或多节点训练。推理仅使用单节点配置。

在开始以下教程之前，请完成中的步骤亚马逊 EKS 安装 (p. 35).

目录

亚马逊 EKS 安装

本指南介绍了如何使用 Amazon Elastic Kubernetes 服务 (Amazon EKS) 设置深度学习环境AmazonDeep Learning Containers。利用 Amazon EKS，您可以借助 Kubernetes 容器扩展用于多节点训练和推理的生产就绪型环境。

如果您尚不熟悉 Kubernetes 或 Amazon EKS，也没关系。本指南和相关的 Amazon EKS 文档展示了如何使用 Kubernetes 工具系列。本指南假定您已熟悉您的深度学习框架的多节点实施以及如何在容器外部设置推理服务器。

Amazon EKS 上的深度学习容器设置包含一个或多个容器（形成集成）。您可以有专用集群类型，如用于训练的集群和用于推理的集群。您还可能希望对于您的集群具有不同的实例类型，具体取决于您的深度学习神经网络和模型的需求。

目录

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。提供了使用自定义图像的示例。你可以在不创建自己的情况下尝试开始使用它们。

- 构建AmazonDeep Learning Containers 自定义映像 (p. 86)

Licensing

要使用 GPU 硬件，请使用具有必需的 GPU 驱动程序的 Amazon 系统映像。我们建议结合使用 Amazon EKS 优化 AMI 与 GPU 支持，将使用在本指南的后续步骤使用它们。此 AMI 包括非Amazon需要最终用户许可协议 (EULA) 的软件。您必须在中订阅 EKS 优化的 AMIAmazon Web Services Marketplace并在工作人员节点组中使用 AMI 之前接受 EULA。

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

配置安全设置

要使用 Amazon EKS，您必须具有有权访问多个安全权限的用户账户。这些都是用Amazon Identity and Access Management(IAM) 工具。

1. 按中的步骤操作，创建 IAM 用户或更新现有 IAM 用户在您的Amazon帐户.

2. 获取此用户的凭证。

   a. 访问：https://console.aws.amazon.com/iam/，打开 IAM 控制台。
   b. UNDERUsers，选择用户。
   c. Select Security Credentials.
   d. Select Create access key.
   e. 下载 key pair 或复制信息以供以后使用。

3. 向 IAM 用户添加以下策略。这些策略提供 Amazon EKS、IAM 和 Amazon Elastic Compute Cloud (Amazon EC2) 的必需访问权限。

   a. Select Permissions.
   b. Select Add permissions.
   c. Select Create policy.
   d. 来自Create policy窗口中，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. 将策略命名为EKSFullAccess创建策略。
   g. 导航回Grant permissions窗口。
   h. Select Attach existing policies directly.
   i. 搜索EKSFullAccess，然后选中复选框。
   j. 搜索AWSCloudFormationFullAccess，然后选中复选框。
   k. 搜索AmazonEC2FullAccess，然后选中复选框。
   l. 搜索IAMFullAccess，然后选中复选框。
   m. 搜索AmazonEC2ContainerRegistryReadOnly，然后选中复选框。
   n. 搜索AmazonEKS_CNI_Policy，然后选中复选框。
   o. 搜索AmazonS3FullAccess，然后选中复选框。
   p. 接受更改。

网关节点

要设置 Amazon EKS 集群，请使用开源工具，eksctl. 我们建议您结合使用 Amazon EC2 实例与深度学习基础 AMI (Ubuntu) 来分配和控制您的集群。您可以在计算机或已运行的 Amazon EC2 实例上本地运行这些工具。但是，为了简化本指南，我们假定您使用的是带 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，您可以在此同一区域中启动您的网关节点。你可以快速关注10 分钟教程该命令指导您启动 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AmazonCLI。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. 通过运行以下命令安装 eksctl。有关 eksctl 的更多信息，请参阅eksctl 文档.

```
$ curl --silent \
--location "https://github.com/weaveworks/eksctl/releases/download/latest_release/
eksctl_$(uname -s)_amd64.tar.gz" \
| tar xz -C /tmp
$ sudo mv /tmp/eksctl /usr/local/bin
```

4. 安装kubectl按照安装 kubectl指南。

   Note

   您必须使用kubectl与 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，1.18kubectl客户端使用 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 aws-iam-AM 身份验证器的更多信息，请参阅安装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Amazon访问密钥，然后Amazon您在 IAM 控制台中访问的秘密访问密钥并将这些密钥粘贴到中的提示符中。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。您必须在运行它之前进行以下修改。

   - name是您将用于管理集群的内容。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是亚马逊 EKS kubernetes 版本。有关受支持的 Amazon EKS 版本，请参阅可用的亚马逊 EKS Kubernetes 版本.
   - nodes是您希望集群中包含的实例的数量。在本示例中，我们将从三个节点开始。
   - node-type指的是实例类。如果您已知道最适合您的情况的实例类，则可以选择不同的实例类。
   - timeout和*ssh-access *可以单独放置。
   - ssh-public-key是要用于登录工作线程节点的密钥的名称。使用已使用的安全密钥或创建新的安全密钥，但请务必使用已为您使用的区域分配的密钥交换出 ssh-public-key。注意：您只需提供 Amazon EC2 控制台的 “密钥对” 部分中看到的密钥名称。
   - region将在其中启动集群的 Amazon EC2 区域。如果您计划使用驻留在某特定区域（非）中的训练数据。*<us-east-1>*）我们建议您使用相同的区域。ssh-public-key 必须具有在此区域中启动实例的权限。

     Note

     本指南的其余部分假定区域是 *<us-east-1>*。

2. 更改命令后，请运行它，然后等待。这对于单节点集群可能需要几分钟时间，如果您选择创建大型集群，则甚至需要更长时间。

```
$ 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，请确保运行适用于 Kubernetes 的 NVIDIA 设备插件使用以下命令在您的集群上使用：

```
$ 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

请参阅上一节关于使用eksctl命令启动 GPU 集群，然后修改`node-type`以使用 CPU 实例类型。

哈瓦那集群

请参阅上述有关使用eksctl命令启动 GPU 集群，然后修改`node-type`与 Habana Gaudi 加速器一起使用实例，例如DL1 实例。

测试您的集群

1. 您可以运行kubectl命令来检查其状态。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。此文件包含您在理想情况下不必修改的一些详细信息，因为将为您生成和更新配置的工具，但最好是在故障排除时引用。

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，它有助于确保找到的默认上下文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

当您想要控制或查询集群时，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，如果您有一个称为 “raining-gpu-1” 的单独的集群，则可以调用get pods命令通过将配置文件作为参数传递，如下所示：

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

值得一提的是，可以不带 kubeconfig 参数运行这一命令，它将报告您的当前主动控制集群上的状态。

```
$ 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，您将经常在活动集群之间切换。因此，您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。

Cleanup

当您使用完集群后，将其删除以避免产生额外成本。

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

要仅删除 pod，请运行以下命令：

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

要重置密钥以便访问集群，请运行以下命令：

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

删除nodegroup要连接到集群，请运行以下命令：

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

附加nodegroup要集群，请运行以下命令：

```
$ 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 Amazon EKS 上的 Deep Learning Containers 的训练和推理，请参阅Amazon EKS 教程 (p. 34).

培训

使用中的步骤创建群集之一。亚马逊 EKS 安装 (p. 35)，您可以使用它来运行训练任务。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。本部分中的主题展示如何使用 Apache MXNet (孵化)、PyTorch、TensorFlow 和 TensorFlow 2 训练示例。

目录

CPU 训练

本部分针对在 CPU 上训练。

有关 Deep Learning Containers 的完整列表，请参阅。Deep Learning Containers 映像 (p. 85). 有关如果您使用的是英特尔数学核心库 (MKL) 的最佳配置设置的提示，请参阅AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87).

目录

Apache MxNet（孵化）CPU 培训

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的内容的说明。此 pod 文件将下载 MXNet 存储库并运行 MNIST 示例。打开vi要么vim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. 使用将 pod 文件分配到集群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. 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。当状态更改为”Completed”，训练任务已完成。

后续步骤

要在 Amazon EKS 上学习基于 CPU 的推理，使用带有 Deep Learning Containers 的 MXNet，请参阅Apache MxNet（孵化）CPU 推理 (p. 61).

TensorFlow CPU 训练

本教程将指导您在单节点 CPU 集群上训练 TensorFlow 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的内容的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。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。

```
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. 使用将 pod 文件分配到集群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。您还可以继续检查”get pods” 以刷新状态。当状态更改为”Completed” 您将知道该训练任务已完成。

后续步骤

要使用带有 Deep Learning Containers 的 TensorFlow 在 Amazon EKS 上学习基于 CPU 的推理，请参阅TensorFlow CPU 推理 (p. 63).

PyTorch CPU 训练

本教程指导您在单节点 CPU 集群上使用 PyTorch 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的内容的说明。此 Pod 文件将下载 PyTorch 存储库并运行 MNIST 示例。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. 使用将 pod 文件分配到集群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. 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。当状态更改为”Completed” 您将知道该训练任务已完成。

使用完集群后，请参阅 EKS 清除以获取有关清除集群的信息。

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 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. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行的内容的说明。此 pod 文件将使用 PyTorch Amazon S3 插件访问示例 Amazon S3 数据集。

   Note

   你的 CPU 集群应该使用c5.12xlarge对于此示例而言，节点或更大。

   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. 使用将 pod 文件分配到集群kubectl.

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

5. 检查状态。任务名称pytorch-s3-plugin这是在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，直到您看到状态更改为 “。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. 检查日志以查看更多详细信息。

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

有关更多信息，请参阅 。PyTorch 的 Amazon S3 插件存储库。

后续步骤

要在 Amazon EKS 上使用 PyTorch 与 Deep Learning Containers 一起学习基于 CPU 的推理，请参阅PyTorch CPU 推理 (p. 66).

GPU 训练

本部分针对在 GPU 上训练。

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 85). 有关如果您使用的是英特尔数学核心库 (MKL) 的最佳配置设置的提示，请参阅AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87).

目录

Apache MxNet（孵化）GPU 培训

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的内容的说明。此 pod 文件将下载 MXNet 存储库并运行 MNIST 示例。打开vi要么vim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. 使用将 pod 文件分配到集群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。多次运行此项，直到您看到状态更改为“。`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. 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。当状态更改为”Completed”，训练任务已完成。

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，使用带有 Deep Learning Containers 的 MxNet，请参阅Apache MxNet（孵化）GPU 推理 (p. 68).

TensorFlow GPU 训练

本教程将指导您在单节点 GPU 集群上训练 TensorFlow 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的内容的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。打开vi要么vim复制并粘贴以下内容。将此文件另存为 `tf.yaml`。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。

```
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. 使用将 pod 文件分配到集群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。多次运行此项，直到您看到状态更改为“。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. 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。当状态更改为”Completed”，训练任务已完成。

后续步骤

要使用带有 Deep Learning Containers 的 TensorFlow 在 Amazon EKS 上学习基于 GPU 的推理，请参阅TensorFlow GPU 推理 (p. 70).

PyTorch GPU 训练

本教程指导您在单节点 GPU 集群上使用 PyTorch 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的内容的说明。此 Pod 文件将下载 PyTorch 存储库并运行 MNIST 示例。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. 使用将 pod 文件分配到集群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。多次运行此项，直到您看到状态更改为 “。`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. 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。当状态更改为”Completed“，训练任务已完成。

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

要在 Amazon EKS 上使用 PyTorch 与 Deep Learning Containers 一起学习基于 GPU 的推理，请参阅PyTorch GPU 推理 (p. 74).

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 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. 为您的集群创建 pod 文件。pod 文件将提供有关集群应运行的内容的说明。此 pod 文件将使用 PyTorch Amazon S3 插件访问示例 Amazon S3 数据集。

   Note

   你的 GPU 集群应该使用`p3.8xlarge`对于此示例而言，节点或更大。

   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. 使用将 pod 文件分配到集群kubectl.

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

5. 检查状态。任务名称pytorch-s3-plugin这是在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，直到您看到状态更改为 “。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. 检查日志以查看更多详细信息。

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

有关更多信息，请参阅 。PyTorch 的 Amazon S3 插件存储库。

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 的完整列表，请参阅Deep Learning Containers 映像 (p. 85).

目录

设置您的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Kubeflow包含所需的组件，例如 PyTorch 运算符、TensorFlow 运算符和 NVIDIA 插件。
- Apache MxNet 和 MPI 运营商。

下载并运行脚本以在群集中安装所需的组件。

```
$ 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 培训

本教程将显示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，您可以使用Kubernetes MxNet 运营商被命名`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任务为您的任务创建一个 pod 文件 (mx_job_dist.yaml)。您需要指定 3 个 jobModes：调度程序、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：可能具有多个服务器，它们存储模型的参数并与工作线程进行通信。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：工作线程节点实际对一批训练样本进行训练。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。
- 您可以从“Always”、“OnFailure”和“Never”之一提供 `restartPolicy`。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. 如需创建 MXJob 模板，请根据您的要求修改以下代码块并将其保存到名为的文件中。`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-表示任务的阶段且将为 “Creating (正在创建)”、“Running (正在运行)”、“CleanUp (清除
状态-提供任务的总体状态且将为 “Running (失败)” 之一。

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
```

Apache MxNet（孵化）与 Horovod 分布式 GPU 培训

本教程将显示如何在使用的多节点 GPU 集群上设置 Apache MXNet (孵化) 模型的分布式训练。Horovod. 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 示例脚本对于 MNIST 模型上的 MxNet。

1. 验证是否已安装 MPiJob 自定义资源。

```
$ 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）以及每个节点具有的 GPU 数量 (gpusPerReplica)。根据您的要求修改以下代码块并将其保存到名为的文件中。`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。作业的名称显示在状态中。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. 清除并重新运行任务：

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

后续步骤

要在 Amazon EKS 上学习基于 GPU 的推理，使用带有 Deep Learning Containers 的 MxNet，请参阅Apache MxNet（孵化）GPU 推理 (p. 68).

TensorFlow 采用 Horovod 分布式 GPU 训练

本教程将显示如何在使用的多节点 GPU 集群上设置 TensorFlow 模型的分布式训练。Horovod. 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 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。

1. 验证是否已安装 MPiJob 自定义资源。

```
$ 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）以及每个节点具有的 GPU 数量 (gpusPerReplica)。根据您的要求修改以下代码块并将其保存到名为的文件中。`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任务。

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

4. 检查状态。作业的名称显示在状态中。如果您正在运行任何其他测试或之前已运行其他测试，它们将显示在此列表中。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. 清除并重新运行任务：

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

后续步骤

要使用带有 Deep Learning Containers 的 TensorFlow 在 Amazon EKS 上学习基于 GPU 的推理，请参阅TensorFlow GPU 推理 (p. 70).

PyTorch 分布式 GPU 培训

本教程将指导您在多节点 GPU 集群上使用 PyTorch 进行分布式训练。它使用 Gloo 作为后端。

1. 验证是否已安装 PyTorch 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
```

该输出应类似于以下内容。

```
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。将其保存到名为的文件中`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>
```

使用完集群后，请参阅 EKS 清除以获取有关清除集群的信息。

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。查看 Amazon EKSPyTorch 的 Amazon S3 插件 GPU 指南入门。

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

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

要在 Amazon EKS 上使用 PyTorch 与 Deep Learning Containers 一起学习基于 GPU 的推理，请参阅PyTorch GPU 推理 (p. 74).

Inference

使用中的步骤创建群集后亚马逊 EKS 安装 (p. 35)，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Amazon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。AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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。亚马逊 EKS 安装 (p. 35).

```
$ 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 (p. 63). 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。亚马逊 EKS 安装 (p. 35).

   ```
   $ 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Amazon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Amazon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。亚马逊 EKS 安装 (p. 35).

```
$ 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 (p. 63). 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 (p. 76).

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 AmazonDeep Learning Containers 英特尔数学核心库 (MKL) 建议 (p. 87)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。亚马逊 EKS 安装 (p. 35).

```
$ 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。亚马逊 EKS 安装 (p. 35).

```
$ 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Amazon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Amazon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 (p. 76).

自定义入口点

对于一些图片，Amazon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AmazonEKS 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AmazonAmazon 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Amazon已使用正确配置凭据`aws configure`或者通过出口Amazon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

Amazon Deep Learning AMIs(dLAMI) 和AmazonDeep Learning 容器(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。本文档详细介绍了 DLAMI 和 DLC 的框架支持策略。有关可用 DLAMI 的列表，请参阅。DLAMI 发行说明. 有关可用 DLC 的列表，请参阅。Deep Learning 容器的发行说明.

支持的框架

请参考下表，查看主动支持哪些框架和版本。请参阅补丁结束来检查多长时间Amazon支持 Origin 框架维护团队积极支持的当前版本。框架和版本在单框架 DLAMI、多框架 DLAMI 或单框架 DLC 中可用。有关特定映像的更多详细信息，请参阅发行说明：

- 单框架 DLAMI发布说明
- 多框架 DLAMI发布说明
- 单框架 DLC发布说明
- 可用的Deep Learning Containers 映像页

Note

在框架版本中k.n.x、k指主要版本，n指的是次要版本，并且x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。

框架	当前版本	支持	CUDA 版本	GitHub GA	补丁结束	提供单一框架 DLAMI	提供多框架 DLAMI	DLC 可用
PyTorch	1.12	是	11.3 对于 SageMaker DLC，DLAMI 为 11.6 和 EC2/ ECS/EKS DLC	02/02/02/02/02	02/02/02/02/02	是2/02/0	是	是
PyTorch	1.11	是	11.3 对于 SageMaker DLC，DLAMI 为 11.5 和 EC2/ ECS/EKS DLC	02/102/102/	02/102/102/	是2/02/10	否	是
PyTorch	1.10.2	是	11.3	10/21/2022	102/21/02/02	是0	否	是
TensorFlow	2.9.1	是	11.2	05/17/2022	05/17/2023	是	是	是
TensorFlow	2.8.0	是	11.2	02/02/02/02/02	02/02/02/02/02	是2/02/0	否	是
TensorFlow	2.7.1	是	11.2	114/04/04/04/04	14/04/04/04/04	是0	否	是

常见问题

哪些框架版本获得安全补丁？

如果框架版本被标记支持中的支持的框架 (p. 80)表，它会获得安全补丁。

图像有哪些作用Amazon发布新框架版本时发布？

我们在发布新版本后不久就发布了新的 DLAMI 和 DLC TensorFlow 和 PyTorch被释放。这包括主要版本、主要次要版本和 major-minor-patch 框架的版本。我们还会在有新版本的驱动程序和库可用时更新映像。有关映像维护的更多信息，请参阅我的框架版本的主动支持何时结束？ (p. 82)

哪些图像是新的 SageMaker/Amazon功能？

新功能通常在最新版本的 DLAMI 和 DLC 中发布 PyTorch 和 TensorFlow. 有关新映像的详细信息，请参阅发行说明 SageMaker 要么Amazon功能。有关可用 DLAMI 的列表，请参阅。DLAMI 发行说明. 有关可用 DLC 的列表，请参阅。的发行说明AmazonDeep Learning 容器. 有关映像维护的更多信息，请参阅我的框架版本的主动支持何时结束？ (p. 82)

如何在 “支持的框架” 表中定义当前版本？

的当前版本支持的框架 (p. 80)table 指的是最新的框架版本Amazon在上可用 GitHub. 每个最新版本都包括对 DLAMI 或 DLC 中的驱动程序、库和相关软件包的更新。有关映像维护的信息，请参阅我的框架版本的主动支持何时结束？ (p. 82)

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支持的框架 (p. 80)表，则可能没有最新的驱动程序、库和相关软件包。有关 up-to-date 版本，我们建议您使用您选择的最新 DLAMI 或 DLC 升级到其中一个受支持的框架。有关可用 DLAMI 的列表，请参阅。DLAMI 发行说明. 有关可用 DLC 的列表，请参阅。的发行说明AmazonDeep Learning 容器.

DLAMI 或 DLC 是否支持以前版本的 TensorFlow?

否。 我们支持每个框架的最新主要版本的最新补丁版本，自其初始版本起365天后发布 GitHub 如上所述支持的框架 (p. 80)表。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？ (p. 81)

如何找到支持的框架版本的最新补丁映像？

要使用带有最新框架版本的 DLAMI，请检索DLAMI然后用它来启动 DLAMIEC2 控制台. 适用于示例 AmazonCLI 命令来检索Amazon Deep Learning AMIID，请参阅深度学习框架的部分Amazon Deep Learning AMI目录.Amazon CLI AMI ID 查询也包含在单框架 DLAMI 发行说明. 您选择的框架版本必须贴上标签支持中的支持的框架 (p. 80)表。

要使用最新框架版本的 DLC，请浏览DLC GitHub发布标签找到您选择的示例映像 URI，并使用它来提取最新的可用的 Docker 镜像。您选择的框架版本必须贴上标签支持中的支持的框架 (p. 80)表。

新映像发布的频率如何？

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映像。我们监控新修补的框架版本（例如 TensorFlow 2.9 TensorFlow 2.9.1）和新的次要发行版本（例如 TensorFlow 2.9 TensorFlow 2.10），并尽早提供它们。当的现有版本 TensorFlow 与新版本的 CUDA 一起发布，我们为该版本发布了新的 DLAMI 和 DLC TensorFlow 支持新的 CUDA 版本。

我的工作负载运行时，我的实例是否会在原地进行修补？

否。 DLAMI 和 DLC 的补丁更新不是 “就地” 更新。

对于 DLAMI，您必须打开新的 EC2 实例，迁移工作负载和脚本，然后关闭以前的实例。

对于 DLC，您必须删除实例上的现有映像，并在不终止实例的情况下拉取最新的容器映像。

当有新的修补或更新的框架版本可用时会发生什么？

定期查看发行说明页面以了解您的映像。我们鼓励您在新的修补或更新的框架可用时升级到它们。有关可用 DLAMI 的列表，请参阅。DLAMI 发行说明. 有关可用 DLC 的列表，请参阅。的发行说明AmazonDeep Learning 容器.

依赖关系是否在不更改框架版本的情况下更新？

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，我们会使用其他版本创建一个映像。请务必检查DLAMI 发行说明和的发行说明Amazon Deep Learning AMI以获取更新的依赖项信息。

我的框架版本的主动支持何时结束？

DLAMI 和 DLC 镜像是不可变的。一旦它们被创建，它们就不会改变。对框架版本的主动支持终止有四个主要原因：

- 框架版本（补丁）升级 (p. 83)
- Amazon安全补丁 (p. 83)
- 补丁结束日期（过期） (p. 83)
- 依赖关系 end-of-support (p. 83)

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 DLAMI 或 DLC 的发行说明页面，并在进行更改时进行升级。

框架版本（补丁）升级

如果你有一个 DLAMI 或 DLC 工作负载基于 TensorFlow 2.7.0 TensorFlow版本 2.7.1 GitHub，那么Amazon发布新的 DLAMI 和 DLC TensorFlow 2.7.1 之前的 2.7.0 版本的映像一旦新映像具有 TensorFlow 2.7.1 已发布。DLAMI 或 DLC 带有 TensorFlow 2.7.0 没有收到更多修补程序。的 DLAMI 和 DLC 发行说明页面 TensorFlow 然后使用最新信息更新 2.7。每个次要补丁都没有单独的发行说明页面。

由于补丁程序升级而创建的新 DLAMI 被指定为新的AMI. 由于修补程序升级而创建的新 DLC 被指定为已更新发布标签. 如果更改不向后兼容，则标记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 v1.2）。

Amazon安全补丁

如果您有基于映像的工作负载 TensorFlow 2.7.0Amazon制作安全补丁，然后发布新版本的 DLAMI 或 DLC TensorFlow 2.7.0。以前的版本的图片 TensorFlow 2.7.0 不再处于主动维护状态。有关更多信息，请参阅 。我的工作负载运行时，我的实例是否会在原地进行修补？ (p. 82)有关查找最新 DLAMI 或 DLC 的步骤，请参阅如何找到支持的框架版本的最新补丁映像？ (p. 82)

由于补丁程序升级而创建的新 DLAMI 被指定为新的AMI. 由于修补程序升级而创建的新 DLC 被指定为已更新发布标签. 如果更改不向后兼容，则标记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 v1.2）。

补丁结束日期（过期）

DLAMI 和 DLC 的补丁截止日期在 365 天后 GitHub 发布日期。

适用于多框架 dLAMI，当其中一个框架版本更新时，需要使用更新版本的新 DLAMI。使用旧框架版本的 DLAMI 不再主动维护。

Important

当有重大框架更新时，我们会例外。例如。 TensorFlow 1.15 的更新 TensorFlow 2.0，那么我们将继续支持最新版本的 TensorFlow 1.15 的有效期为从该日起的两年 GitHub 发布或原始框架维护团队放弃支持六个月后（以较早的日期为准）。

依赖关系 end-of-support

如果您正在运行工作负载 TensorFlow 2.7.0 使用 Python 3.6 的 DLAMI 或 DLC 映像以及该版本的 Python 标记为 end-of-support，那么所有基于 Python 3.6 的 DLAMI 和 DLC 映像都将不再主动维护。同样，如果像 Ubuntu 16.04 这样的操作系统版本被标记为 end-of-support，那么所有依赖于 Ubuntu 16.04 的 DLAMI 和 DLC 映像都将不再主动维护。

框架版本不再主动维护的镜像会被修补吗？

否。 不再主动维护的映像将不会有新版本。

如何使用旧的框架版本？

要将 DLAMI 与较早的框架版本一起使用，请检索DLAMI然后用它来启动 DLAMIEC2 控制台. 适用于Amazon用于检索 AMI ID 的 CLI 命令，请参阅深度学习框架的部分Amazon深度学习 AMI.Amazon CLI AMI ID 查询也包含在单框架 DLAMI 发行说明.

要将 DLC 与较早的框架版本一起使用，请浏览DLC GitHub发布标签找到你选择的镜像 URI 并用它来拉取 docker 镜像。

我如何入住 up-to-date 框架及其版本的支持变更？

留 up-to-date 使用 DLAMI 和 DLC 框架和版本使用支持的框架 (p. 80)表，DLAMI 发行说明，DLC 发行说明，以及可用的Deep Learning Containers 映像页.

我需要商业许可证才能使用 Anaconda 存储库吗？

Anaconda 为某些用户转向商业许可模式。积极维护的 DLAMI 和 DLC 已迁移到公开可用的开源版本的 Conda (康达·福格) 来自 Anaconda 频道。

> Warning
>
> 如果你正在积极使用 Anaconda 在不再主动维护的 DLC 中安装和管理你的软件包及其依赖项，你有责任遵守Anaconda 存储库，如果您确定这些条款适用于您。或者，您可以迁移到当前支持的其中一个 DLC，该文件位于支持的框架 (p. 80)表或者你可以使用 conda-forge 作为源代码来安装软件包。

Deep Learning Containers 映像

AmazonDeep 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 (p. 2)
- 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 教程 (p. 12)
- 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 教程 (p. 34)
- 有关 Deep Learning Containers 中的安全性的信息，请参阅中的安全性AmazonDeep Learning 容器 (p. 91)
- 有关最新 Deep Learning Containers er 发布说明的列表，请参阅Deeeep Learning 容器 (p. 97)

Deep Learning Containers 资源

以下主题介绍了其他AmazonDeep Learning Containers 资源。

目录

构建AmazonDeep 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Amazon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
```

AmazonDeep 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 AmazonEC2 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
```

中的安全性AmazonDeep Learning 容器

Amazon 十分重视云安全性。作为 Amazon 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是Amazon和您的共同责任。责任共担模型将其描述为云的安全性和云中的安全性：

- 云的安全性 – Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础设施。Amazon 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 Amazon Compliance Programs 的一部分. 要了解适用于Deep Learning Containers 的合规性计划，请参阅Amazon合规性计划范围内的服务.
- 云中的安全性 - 您的责任由您使用的 Amazon 服务决定。您还需要对其它因素负责，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。

此文档将帮助您了解如何在使用Deep Learning Containers 时应用责任共担模式。以下主题说明如何配置 Dep Learning Containers 以实现您的安全性和合规性目标。您还将了解如何使用其他Amazon服务，以帮助您监控和保护Deep Learning Containers 资源。

有关更多信息，请参阅 。Amazon EC2 中的安全性、Amazon ECS 中的安全性、Amazon EKS 中的安全性, 和Amazon 的安全性 SageMaker.

主题

中的数据保护AmazonDeep Learning 容器

这些区域有：Amazon 责任共担模式适用于中的数据保护AmazonDeep Learning Containers。如该模式中所述，Amazon 负责保护运行所有 Amazon Web Services 云 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。此内容包括您所使用的 Amazon Web Services 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。

出于数据保护目的，我们建议您保护 Amazon Web Services 账户 凭证并使用 Amazon Identity and Access Management (IAM) 设置单独的用户账户。这仅向每个用户授予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您的数据：

- 对每个账户使用 Multi-Factor Authentication (MFA)。
- 使用 SSL/TLS 与 Amazon 资源进行通信。建议使用 TLS 1.2 或更高版本。
- 使用 Amazon CloudTrail 设置 API 和用户活动日志记录。
- 使用 Amazon 加密解决方案以及 Amazon 服务中的所有默认安全控制。

- 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 Simple Storage Service（Amazon S3）中的个人数据。
- 如果在通过命令行界面或 API 访问 Amazon 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 终端节点。有关可用的 FIPS 终端节点的更多信息，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（例如您客户的电子邮件地址）放入标签或自由格式字段（例如名称字段）。这包括使用Dep Learning Containers 或其他Amazon服务使用控制台、API、Amazon CLI，或者Amazon开发工具包。您在用于名称的标签或自由格式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 URL，我们强烈建议您不要在 URL 中包含凭证信息来验证您对该服务器的请求。

中的 Identity and Access ManagementAmazonDeep Learning 容器

Amazon Identity and Access Management (IAM) 是一项 Amazon Web Service，可以帮助管理员安全地控制对 Amazon 资源的访问。IAM 管理员控制谁可以成为身份验证（已登录）和授权（具有权限）以使用Deep Learning Containers 资源。IAM 是一项无需额外费用即可使用的 Amazon Web Service。

有关Identity and Access Management ent 的更多信息，请参阅适用于 Amazon EC2 的 Identity an、适用于 Amazon ECS 的 Identity an、适用于 Amazon EKS 的 Identity, 和Amazon Identity and Ac SageMaker.

主题

使用身份进行身份验证

身份验证是您使用身份凭证登录 Amazon 的方法。有关使用Amazon Web Services Management Console 登录的更多信息，请参阅 IAM 用户指南中的以 IAM 用户或根用户身份登录Amazon Web Services Management Console。

您必须作为Amazon Web Services 账户根用户、IAM 用户或代入 IAM 角色以进行身份验证（登录到 Amazon）。您还可以使用公司的单一登录身份验证方法，甚至使用 Google 或 Facebook 登录。在这些情况下，您的管理员以前使用 IAM 角色设置了联合身份验证。在您使用来自其它公司的凭证访问 Amazon 时，您间接地代入了角色。

要直接登录到Amazon Web Services Management Console，请将密码与根用户电子邮件地址或 IAM 用户名一起使用。您可以使用根用户或 IAM 用户访问密钥以编程方式访问 Amazon。Amazon 提供了 SDK 和命令行工具，可使用您的凭证对您的请求进行加密签名。如果您不使用 Amazon 工具，则必须自行对请求签名。使用 Signature Version 4（用于对入站 API 请求进行验证的协议）完成此操作。有关验证请求的更多信息，请参阅《Amazon 一般参考》中的 Signature Version 4 签名流程。

无论使用何种身份验证方法，您可能还需要提供其它安全信息。例如，Amazon 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《IAM 用户指南》中的在 Amazon 中使用多重身份验证 (MFA)。

Amazon Web Services 账户根用户

当您首次创建 Amazon Web Services 账户时，最初使用的是一个对账户中所有 Amazon Web Services和资源有完全访问权限的单点登录身份。此身份称为Amazon Web Services 账户根用户，使用您创建账户时所用

的电子邮件地址和密码登录，即可获得该身份。强烈建议您不使用根用户执行日常任务，即使是管理任务。相反，请遵循仅使用根用户创建您的第一个 IAM 用户的最佳实践。然后请妥善保存根用户凭证，仅用它们执行少数账户和服务管理任务。

IAM 用户和组

IAM 用户是Amazon Web Services 账户内对某个人员或应用程序具有特定权限的一个身份。IAM 用户可能具有长期凭证，例如用户名和密码或一组访问密钥。要了解如何生成访问密钥，请参阅 IAM 用户指南 中的管理 IAM 用户的访问密钥。为 IAM 用户生成访问密钥时，请确保查看并安全保存密钥对。您以后无法找回秘密访问密钥，而是必须生成新的访问密钥对。

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

IAM 角色是Amazon Web Services 账户中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。您可以通过切换角色，在 Amazon Web Services Management Console 中暂时代入 IAM 角色。您可以调用 Amazon CLI 或 Amazon API 操作或使用自定义 URL 以代入角色。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：

- 临时 IAM 用户权限 – IAM 用户可以代入 IAM 角色，以暂时获得不同的权限以执行特定的任务。
- 联合身份用户访问 – 您可以不创建 IAM 用户，而是使用来自 Amazon Directory Service、您的企业用户目录或 Web 身份提供商的现有身份。这些用户被称为联合用户。在通过身份提供商请求访问权限时，Amazon 将为联合身份用户分配角色。有关联合身份用户的更多信息，请参阅 IAM 用户指南 中的联合身份用户和角色。
- 跨账户访问 – 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 Amazon Web Services，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 – 某些 Amazon Web Services使用其它 Amazon Web Services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 Simple Storage Service（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。
  - 主体权限 – 当您使用 IAM 用户或角色在 Amazon 中执行操作时，您将被视为主体。策略向主体授予权限。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中触发另一个操作。在这种情况下，您必须具有执行这两个操作的权限。要查看某个操作是否需要策略中的其他相关操作，请参阅服务授权参考。
  - 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担任的 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 Amazon Web Service 委派权限的角色。
  - 服务相关角色 – 服务相关角色是与 Amazon Web Service关联的一种服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon 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 Amazon CLI 或 Amazon API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 Amazon 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到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

您将创建策略并将其附加到 IAM 身份或Amazon资源，以便控制Amazon中的访问。策略是Amazon中的对象；在与标识或资源相关联时，策略定义它们的权限。您可以通过 root 用户或 IAM 用户身份登录，也可以代入 IAM 角色。随后，当您提出请求时，Amazon会评估相关的基于身份或基于资源的策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 Amazon 中存储为 JSON 文档。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 Amazon JSON 策略来指定谁有权访问什么内容。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。

每个 IAM 实体（用户或角色）最初没有任何权限。换言之，预设情况下，用户什么都不能做，甚至不能更改他们自己的密码。要为用户授予执行某些操作的权限，管理员必须将权限策略附加到用户。或者，管理员可以将用户添加到具有预期权限的组中。当管理员为某个组授予访问权限时，该组内的全部用户都会获得这些访问权限。

IAM policy 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。具有该策略的用户可以从 Amazon Web Services Management Console、Amazon CLI 或 Amazon 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 policy。

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。托管式策略是可以附加到Amazon Web Services 账户中的多个用户、组和角色的独立策略。托管式策略包括 Amazon 托管式策略和客户托管式策略。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。主体可以包括账户、用户、角色、联合身份用户或 Amazon Web Services。

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来自 IAM 的 Amazon 托管式策略。

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。

Simple Storage Service（Amazon S3）、Amazon WAF 和 Amazon VPC 是支持 ACL 的服务示例。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

Amazon 支持额外的、不太常用的策略类型。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的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，指定了组织或组织单位 (OU) 在 Amazon Organizations 中的最大权限。Amazon Organizations 服务可以分组和集中管理您的企业拥有的多个 Amazon Web Services 账户 账户。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 限制成员账户中实体的权限，包括每个 Amazon Web Services 账户 根用户。有关 Organizations 和 SCP 的更多信息，请参阅 Amazon 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身份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 Amazon 如何确定在涉及多种策略类型时是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

IAM 与 Amazon EMR 结合使用

您可以使用Amazon Identity and Access Management使用 Amazon EMR 来定义用户，Amazon资源、组、角色和策略。你也可以控制哪个Amazon这些用户和角色可以访问的服务。

有关 IAmazon EMR 的更多信息，请参阅Amazon适用于 Amazon EMR 的 IdIdentity and Access Management ity and A.

中的日志记录和监控AmazonDeep Learning 容器

您的AmazonDeep Learning Containers 不附带监控实用程序。有关监控的信息，请参阅GPU 监控和优化、监控Amazon EC2、监控 Amazon ECS、监控 Amazon E, 和监控Amazon SageMaker.

使用情况跟踪

以下Deep Learning Containers 包含的代码允许Amazon以收集用于容器的实例类型、框架、框架版本、容器类型和 Python 版本。不会收集或保留有关容器中使用的命令的信息。不会收集或保留有关容器的其他信息。

- TensorFlow 1.15
- TensorFlow 2.0
- TensorFlow 2.1
- PyTorch 1.2
- PyTorch 1.3.1
- MXNet 1.6

要选择退出使用情况跟踪，请使用自定义入口点禁用对以下服务的调用：

- Amazon EC2 自定义入口点
- Amazon ECS 自定义入口点
- Amazon EKS 自定义入口点

选择退出使用情况跟踪 TensorFlow （版本 >=1.15 ）， PyTorch （版本 >= 1.5 ）和 MXNet （版本 >= 1.6 ）容器，你还需要设置OPT_OUT_TRACKING环境变量。

```
OPT_OUT_TRACKING=true
```

的合规性验证AmazonDeep Learning 容器

作为多个项目的一部分，第三方审计员将评估服务的安全性和合规性Amazon合规性计划。有关支持的合规性计划的信息，请参阅Amazon EC2 的合规性验、Amazon ECS 的合规性验、Amazon EKS 的合规性，和Amazon 合规性验证 SageMaker.

有关特定合规性计划范围内的 Amazon 服务列表，请参阅合规性计划范围内的 Amazon 服务。有关常规信息，请参阅Amazon合规性计划。

您可以使用 Amazon Artifact 下载第三方审计报告。有关更多信息，请参阅下载 Amazon Artifact 中的报告。

您在使用Deep Learning Containers 时的合规性责任由您的数据的敏感性、您公司的合规性目标以及适用的法律法规决定。Amazon提供了以下资源来帮助实现合规性：

- 安全性与合规性 Quick Start 指南安全性与合规性 Quick Start 指南 - 这些部署指南讨论了架构注意事项，并提供了在 Amazon 上部署基于安全性和合规性的基准环境的步骤。
- Amazon合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- Amazon Config 开发人员指南中的使用规则评估资源 – 此 Amazon Config 服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- Amazon Security Hub – 此 Amazon 服务提供了 Amazon 中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实践。

中的故障恢复能力AmazonDeep Learning 容器

Amazon全球基础设施围绕Amazon区域和可用区构建。Amazon区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。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故障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 Amazon 区域和可用区的更多信息，请参阅 Amazon 全球基础设施。

有关帮助支持数据弹性和备份需求的功能的信息，请参阅Amazon EC2 中的恢复功能、Amazon EKS 中的恢复能力, 和Amazon 的恢复能力 SageMaker.

中的基础设施安全性AmazonDeep Learning 容器

的基础架构安全性AmazonDeep Learning Containers 由 Amazon EC2、亚马逊 ECS、亚马逊 EKS 或 SageMaker. 有关更多信息，请参阅 。Amazon EC2 中的基础设施安全性、Amazon ECS 中的基础设施安全性、Amazon EKS 中的基础设施安全性, 和Amazon 基础设施安全性 SageMaker.

Deeeep Learning 容器

请查看最新发行说明AmazonDeep Learning Containers 专为特定的机器学习框架、基础架构和Amazon服务。

Note

从 MXNet 1.9 开始 PyTorch 1.10 和 Tensorflow 2.7、CPU 和 GPU Deep Learning Containers 以任一形式发布E3要么SageMaker映像。Amazon EC2、亚马逊 ECS 和亚马逊 EKS 支持 E3 映像，而 SageMaker 亚马逊支持图片 SageMaker 并包含其他要支持的软件包 SageMaker 工作负载。

Deep Learning Containers

TensorFlow

- AmazonDeep Learning Containers TensorFlow 2.9.0（推断开了 SageMaker): 2022 年 6 月 18 日
- AmazonDeep Learning Containers TensorFlow 2.9.0（E3 上的 Inference Deep Learning 2022 年 6 月 18 日
- AmazonDeep Learning Containers TensorFlow 2.9.1（关于的培训 SageMaker): 2022 年 6 月 13 日
- AmazonDeep Learning Containers TensorFlow 2.9.0 (E3 上的 E3 培训): 2.9.0 2022 年 5 月 19 日
- AmazonDeep Learning Containers TensorFlow 2.8.0 2022 年 3 月 22 日
- AmazonDeep Learning Containers TensorFlow 2.8.0SageMaker): 2022 年 3 月 22 日
- AmazonDeep Learning Containers TensorFlow 2.7.0 版：2021 年 12 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.7.0SageMaker): 2022 年 3 月 22 日
- AmazonDeep Learning Containers TensorFlow 2.6.0 .0 2021 年 9 月 24 日
- AmazonDeep Learning Containers TensorFlow 2.5.0 .0 2021 年 7 月 0日
- AmazonDeep Learning Containers TensorFlow 2.4.1：2021 年 3 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.3.2：2021 年 3 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.2 .2 .2. 2021 年 3 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.1.3 .0 2021 年 3 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.0.4 美元 2021 年 3 月 15 日
- AmazonDeep Learning Containers TensorFlow 2.3.1 2020 年 10 月 15 日
- AmazonDeeep LElastic Inference 容器 TensorFlow 2.0.0 .0 2020 年 8 月 31 日
- AmazonDeeep LElastic Inference 容器 TensorFlow 1.15.0 .0 .0 2020 年 8 月 31 日
- AmazonDeep Learning Containers TensorFlow 2.3.0 .0 2020 年 8 月 7 日
- AmazonDeep Learning Containers TensorFlow 版本 2.2.0 with CUDA 10.2 2020 年 7 月 24 日
- AmazonDeep Learning Containers TensorFlow 2.2.0 2020 年 7 月 20 日
- Amazon适用于 Tensorflow 的Deep Learning Containers 1.15.3 2020 年 7 月 29 日
- AmazonDeep Learning Containers TensorFlow 2.1.1 .0 2020 年 6 月 25 日
- AmazonDeep Learning Containers TensorFlow 2.0.2 .2 美元 2020 年 6 月 19 日
- AmazonDeep Learning Containers TensorFlow 2.2.0 2020 年 5 月 20 日
- AmazonTensorFlow 2.1.1.0 DeeDeep Learning Containers 2020 年 3 月 19 日
- AmazonDeep Learning Containers TensorFlow 2.0 2020 年 2 月 26 日
- AmazonDeep Learning Containers TensorFlow 1.15 支持 python-3.7: 2020 年 5 月 6 日
- Amazon适用于 Tensorflow 的Deep Learning Containers 1.15.2 2020 年 3 月 19 日

PyTorch

- AmazonDeep Learning Containers PyTorch 1.12.0SageMaker): 2022 年 7 月 26 日
- AmazonDeep Learning Containers PyTorch 1.12.0 版：2022 年 7 月 0日
- AmazonDeep Learning Containers PyTorch 1.11.0SageMaker): 2022 年 5 月 6 日
- AmazonDeep Learning Containers PyTorch 1.11.0 2022 年 4 月 14 日
- AmazonDeep Learning Containers PyTorch 1.10.0 2021 年 11 月 3 日
- AmazonDeep Learning Containers PyTorch 1.10.2SageMaker): 2022 年 4 月 14 日
- AmazonDeep Learning Containers PyTorch 1.9.0 2021 年 8 月 30 日
- AmazonDeep Learning Containers PyTorch 1.8.0 2021 年 3 月 16 日
- AmazonDeep Learning Containers PyTorch 1.7.1 2021 年 3 月 15 日
- AmazonDeep Learning Containers PyTorch 1.6.0 2020 年 12 月 9 日
- AmazonDeeep LElastic Inference 容器 PyTorch 1.3.1：2020 年 8 月 31 日
- AmazonDeep Learning Containers PyTorch 1.6.0 版：2020 年 8 月 3 日
- AmazonDeep Learning Containers PyTorch 1.5.1 .1 2020 年 6 月 19 日
- AmazonDeep Learning Containers PyTorch 1.4 2020 年 6 月 6 日
- AmazonDeep Learning Containers PyTorch 1.5.0 2020 年 5 月
- AmazonDeep Learning Containers PyTorch 1.4 2020 年 4 月 03 日

Amazon由 Apache MXNet 提供支持的 MX

- Amazon适用于 MXNet 1.9.0 (E3) 的Deep Learning Containers: 2022 年 1 月 31 日
- Amazon适用于 MXNet 1.9.0 的Deep Learning Containers (SageMaker): 2022 年 1 月 31 日
- AmazonMXNDeep Learning Containers eeeeeeeeee 2020 年 12 月 01 日
- AmazonMXNDeep Learning Containers eeeeeeeeee 2020 年 9 月 17 日
- AmazonMXNet 1.5.1 2020 年 8 月 31 日
- AmazonMXNet 1.6.0 Deep Learning Containers：2020 年 7 月 20 日
- AmazonMXNet 1.6.0 Deep Learning Containers：2020 年 4 月 03 日

Deep Learning Containers

TensorFlow

- AmazonDeeep Learning 容器 TensorFlow 2.7.0 版：2021 年 12 月

PyTorch

- AmazonDeeep Learning 容器 PyTorch 1.10.0 2021 年 12 月

Habana Deep Learning Containers

TensorFlow

- AmazonHabana SynapSearners TensorFlow 2.9.1 .0 2022 年 7 月 25 日
- AmazonHabana SynapSearners TensorFlow 2.8.0 2022 年 6 月 23 日
- AmazonHabana SynapSearners TensorFlow 2.8.0 2022 年 3 月 24 日
- AmazonHabana SynapSearners TensorFlow 2.7.0 版 2022 年 2 月 15 日

- Amazon适用于 Habana SynapseAI 的Deep Learning Containers 0.15.4 TensorFlow 2.5.0 .0 2021 年 10 月 26 日

PyTorch

- AmazonHabana Deeeep Learning 容器 1.5.0 PyTorch 1.11.0 2022 年 7 月 25 日
- AmazonHabana SynapSearners PyTorch 1.10.2 美元 2022 年 6 月 23 日
- AmazonHabana SynapSearners PyTorch 1.10.1 .1 美元 2022 年 3 月 24 日
- AmazonHabana SynapSearners PyTorch 1.10.0 2022 年 2 月 15 日
- Amazon适用于 Habana SynapseAI 的Deep Learning Containers 0.15.4 PyTorch 1.7.1：2021 年 10 月 26 日

Deep Learning Containers 开发人员指南文档历史记录

下表介绍此版本的 Deep Learning Containers als 的文档。

- API 版本：最新
- 最近文档更新时间： 2020 年 2 月 26 日

update-history-change	更新-历史记录-描述	更新-历史记录-日期
Apache MXNet 与霍罗伍德 (p. 1)	Apache MXNet 教程添加到开发人员指南中。	2020 年 2 月 26 日
Deep Learning Containers 开发人员指南启动 (p. 1)	在开发人员指南中添加了 Deep Learning Containers als 设置和教程。	2020 年 2 月 17 日

Amazon词汇表

For the latest Amazon terminology, see the Amazon glossary in the Amazon General Reference.

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。