亚马逊云科技

开发人员指南

Amazon 深度学习容器

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 深度学习容器: 开发人员指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 中国的 Amazon Web Services 服务入门 (PDF)。

Table of Contents

什么是 Amazon 深度学习容器？

Amazon Deep Learning Containers 是预先构建的 Docker 镜像，可以更轻松地运行流行的深度学习框架和工具。 Amazon它们为托管在 Amazon 基础设施上的深度学习应用程序提供一致 up-to-date、安全和优化的运行时环境。要开始使用，请参阅 Dee Amazon p Learning Containers 入门。

主要功能

预安装的深度学习框架

Amazon Deep Learning Containers 包括领先的深度学习框架的预安装和配置版本，例如 TensorFlow 和。 PyTorch这样就无需从头开始构建和维护自己的 Docker 镜像。

硬件加速

Amazon Deep Learning Conta CPU iners 针对基于深度学习、GPU加速学习和 Amazon 基于硅的深度学习进行了优化。它们支持CUDADNN、cu 和其他必要的库，以利用GPU基于 Amazon 的EC2实例以及由 Graviton 和 Amazon Tra CPUs inium 等 Amazon设计的芯片以及英特尔的 Habana-Gaudi 处理器的强大功能。GPUs

Amazon 服务集成

Amazon Deep Learning Containers 可与各种 Amazon 服务无缝集成 SageMaker，包括亚马逊弹性容器服务 (ECS)、亚马逊 Elastic Kubernetes Service (EKS)、亚马逊和。EC2 Amazon ParallelCluster这样可以轻松地在 Amazon 基础架构上部署和运行深度学习模型和应用程序。

安全且定期更新

Amazon 定期维护和更新 Dee Amazon p Learning Containers，以确保您可以访问最新版本的深度学习框架和依赖关系。这有助于保持 Amazon基于您的深度学习环境的安全 up-to-date，并且无需自己管理安全补丁和更新的开销。使用最新的安全补丁更新深度学习容器可能是一项资源密集型任务，但是 Dee Amazon p Learning Containers 通过提供定期的自动更新来消除这种负担。这样可以确保您的深度学习环境保持安全和最新，而无需您进行大量手动操作。通过自动化更新过程，Dee Amazon p Learning Containers 使您可以专注于开发深度学习模型和应用程序，而不必担心底层基础设施和安全维护，这可以提高团队的工作效率，并使您能够在托管的项目中更有效地利用最新的深度学习功能。Amazon

应用场景

Amazon Deep Learning Containers 在以下 Amazon基于深度学习的场景中特别有用：

模型训练

使用 Dee Amazon p Learning Containers 在CPU基于、GPU加速或 Amazon 硅驱动的 A EC2 mazon 实例上训练您的深度学习模型，或者在 Hyperpod 上利用多节点训练 Amazon ParallelCluster 。 SageMaker

模型部署

使用 Dee Amazon p Learning Containers 部署经过训练的模型，以便在上面进行可扩展、可用于生产的推理 Amazon，例如通过。 SageMaker

实验和原型设计

Amazon 使用预先配置的容器快速启动深度学习开发环境。 Amazon Deep Learning Containers 是 SageMaker Studio 中笔记本的默认选项，可以轻松开始实验和原型设计。

持续集成和交付

将容器集成到 Amazon基于您的 CI/CD 管道（例如使用 Amazon 或 A ECS mazon 的管道）中，以实现一致EKS、自动化的深度学习工作负载。

Dee Amazon p Learning Containers 入门

以下各节介绍如何使用 Deep Learning Containers 在 Amazon 基础架构上运行每个框架中的示例代码。

使用案例

- 有关将 Deep Learning Containers 与配合使用的信息 SageMaker，请参阅使用自己的算法或模型和 SageMaker 文档。
- 要了解如何在开启的情况下使用 Deep Learning C SageMaker HyperPod ontainersEKS，请参阅使用 Amazon 编排 SageMaker HyperPod 集群。EKS SageMaker

主题

- 自定义深度学习容器 Deep Learnin
- 亚马逊EC2教程
- 亚马逊ECS教程
- 亚马逊EKS教程

自定义深度学习容器 Deep Learnin

我们可以使用 Deep Learning Containers 自定义训练和推理，使用 Docker 文件添加自定义框架、库和包。

示例

在下面的示例 Dockerfile 中，我们将包含许多深度学习模型示例的示例 GitHub 存储库添加到 PyTorch Inference 深度学习容器中。 Amazon

```
# Take base container
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:2.4-cpu-py311-ec2

# Add custom stack of code
RUN git clone https://github.com/aws-samples/deep-learning-models
```

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**
>
> 你可能需要登录才能访问 Deep Learning Containers 图像存储库。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
> ```

亚马逊EC2教程

本节介绍如何在 Deep Learning Containers 上运行训练和推理以EC2使用 PyTorch、和。 TensorFlow

在开始以下教程之前，请完成中的步骤亚马逊EC2设置。

内容

亚马逊EC2设置

在本节中，您将学习如何使用亚马逊弹性计算云设置 Dee Amazon p Learning Containers。

完成以下步骤来配置您的实例：

- 使用以下策略创建 Amazon Identity and Access Management 用户或修改现有用户。您可以在IAM控制台的策略选项卡中按名称搜索它们。
  - 亚马逊 ECS _ FullAccess 政策
  - Amazon EC2ContainerRegistryFullAccess

  有关创建或编辑用户的更多信息，请参阅IAMIAM用户指南中的添加和删除IAM身份权限。
- 启动 Amazon 弹性计算云实例（CPU或GPU），最好是深度学习库AMI。其他AMIs工作，但需要相关的GPU驱动程序。
- 使用连接到您的实例SSH。有关连接的更多信息，请参阅 Amazon EC2 用户指南中的实例连接疑难解答。 。
- 使用安装当前 Amazon CLI版本中的步骤确保您的版本 Amazon CLI 是最新的。
- 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，运行以下命令登录托管 Deep Learning Containers 图像的亚马逊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
```

后续步骤

要了解有关使用 Deep Learning Containers 在亚马逊上EC2进行训练和推理的信息，请参阅亚马逊EC2教程。

训练

本节介绍如何EC2使用 PyTorch 和在 Dee Amazon p Learning Containers for Amazon 上运行训练 TensorFlow。

内容

PyTorch训练

要 PyTorch 从您的 Amazon EC2 实例开始训练，请使用以下命令运行容器。您必须**nvidia-docker**使用GPU图片。

- 对于 CPU

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

- 对于 GPU

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

- 如果你有 docker-ce 版本 19.03 或更高版本，你可以在 docker 中使用--gpus 标志：

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

运行以下命令开始训练。

- 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 使用 NVIDIA Ape GPU x 进行分布式训练

NVIDIAApex 是一款具有用于混合精度和分布式训练的实用程序的 PyTorch 扩展。有关Apex提供的实用程序的更多信息，请访问A NVIDIApex网站。Apex 目前由以下系列的亚马逊EC2实例支持：

要开始使用 NVIDIA Apex 进行分布式训练，请在GPU训练容器的终端中运行以下命令。此示例要求您的 Amazon EC2 实例GPUs上至少有两个，才能运行并行分布式训练。

```
$ 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
```

TensorFlow训练

登录 Amazon EC2 实例后，您可以使用以下命令运行 TensorFlow TensorFlow 2 个容器。您必须nvidia-docker使用GPU图片。

- 对于CPU基于基础的训练，请运行以下命令。

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

- 对于GPU基于基础的训练，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进行导入 TensorFlow。

```
$ python
```

```
>> import tensorflow
```

按 Ctrl+D 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

要在亚马逊上EC2使用 Deep Learning Cont PyTorch ainers 学习推理，请参阅PyTorch推断 。

推理

本节介绍如何使用 PyTorch和在适用于亚马逊弹性计算云的 Dee Amazon p Learning Containers 上运行推理。 TensorFlow

内容

PyTorch推断

1.6 及更高 PyTorch 版本的 Deep Learning Container TorchServe s 用于推理调用。1.5 及更早 PyTorch 版本的 Deep Learning Container `multi-model-server` s 用于推理调用。

PyTorch 1.6 及更高版本

为了运行推理 PyTorch，此示例使用了公共 S3 存储桶在 Imagenet 上预训练的模型。推理是使用 TorchServe提供的。有关更多信息，请参阅这篇关于使用部署 PyTorch 推理的 TorchServe博客。

CPU例如：

```
$ 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
```

举GPU例来说

```
$ 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
```

如果你有 docker-ce 版本 19.03 或更高版本，则可以在启动 Docker 时使用该`--gpus`标志。

配置文件包含在容器中。

服务器启动后，您现在可以使用以下方法从不同的窗口运行推理。

```
$ 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
```

使用完容器后，您可以使用以下方法将其移除。

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

PyTorch 1.5 及更早版本

为了运行推理 PyTorch，此示例使用了公共 S3 存储桶在 Imagenet 上预训练的模型。推理是使用的 multi-model-server，它可以支持任何框架作为后端。有关更多信息，请参阅multi-model-server。

CPU例如：

```
$ docker run -itd --name mms -p 80:8080  -p 8081:8081 <your container image id> \
multi-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
```

举GPU例来说

```
$ nvidia-docker run -itd --name mms -p 80:8080  -p 8081:8081 <your container image id>
 \
multi-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
```

如果你有 docker-ce 版本 19.03 或更高版本，则可以在启动 Docker 时使用该--gpus标志。

配置文件包含在容器中。

服务器启动后，您现在可以使用以下方法从不同的窗口运行推理。

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

使用完容器后，您可以使用以下方法将其移除。

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

TensorFlow推断

为了演示如何使用 Deep Learning Containers 进行推理，此示例使用了一个带有 TensorFlow 2 Serving 的简单半加二模型。我们建议将深度学习基础用AMI于 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
```

使用此处的命令开始使用此模型的 Deep Learning Containers 进行 TensorFlow 服务。与用于训练的 Deep Learning Containers 不同，模型服务在运行容器后立即开始，并作为后台进程运行。

- CPU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 <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
  ```

- GPU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 <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些时间。

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，您可以使用名称附加到容器的输出，如以下命令所示：
>
> ```
> $ docker attach <your docker container name>
> ```
>
> 使用了这个例子tensorflow-inference。

后续步骤

要了解如何在亚马逊的 Deep Learning Containers 中使用自定义入口点ECS，请参阅。自定义入口点

自定义入口点

对于某些图像，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
  ```

亚马逊ECS教程

本节介绍如何ECS使用 PyTorch和在 Dee Amazon p Learning Containers for Amazon 上运行训练和TensorFlow推理。

在开始以下教程之前，请完成中的步骤亚马逊ECS设置。

内容

亚马逊ECS设置

本主题介绍如何使用亚马逊弹性容器服务设置 D Amazon eep Learning Containers。

内容

前提条件

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。有关安装或升级的更多信息 Amazon CLI，请参阅安装 Amazon Command Line Interface。
- 完成 “使用 Amazon 进行设置” 中的步骤ECS。
- 确认您拥有 Amazon ECS 容器实例角色。有关更多信息，请参阅《亚马逊弹性ECS容器服务开发者指南》中的亚马逊容器实例IAM角色。
- 亚马逊 CloudWatch 日志IAM策略已添加到亚马逊ECS容器实例角色中，该角色允许亚马逊ECS向亚马逊发送日志 CloudWatch。有关更多信息，请参阅 Amazon 弹性容器服务开发者指南中的CloudWatch 日志IAM策略。
- 创建新的安全组或更新现有的安全组，以打开所需的推理服务器的端口。
  - 为了进行 TensorFlow 推断，端口 8501 和 8500 对流量开放。TCP

有关更多信息，请参阅 Amazon EC2 安全组。

为 Deep Learning ECS Containers 设置亚马逊

本节介绍如何将亚马逊设置为使用 Deep Lear ECS ning Containers。

> **Important**
>
> 如果您的账户已经创建了 Amazon ECS 服务相关角色，则除非您在此处指定角色，否则该角色将默认用于您的服务。如果您的任务定义使用 awsvpc 网络模式，或者服务配置为使用以下任一功能：服务发现、外部部署控制器、多个目标组或 Elastic Inference 加速器，则需要服务相关角色。如果是这种情况，则不应在此处指定角色。有关更多信息，请参阅《亚马逊ECS开发者指南》ECS中的使用亚马逊服务相关角色。

从您的主机运行以下操作。

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

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

2. 在您的集群中启动一个或多个 Amazon EC2 实例。有关GPU基于基础的工作，请参阅GPUs《亚马逊ECS开发者指南》ECS中的 “在亚马逊上使用”，告知您的实例类型选择。如果您选择GPU实例类型，请务必选择亚马逊ECSGPU优化的AMI实例类型。对于CPU基于基础的工作，你可以使用ECS经过优化的AMIs亚马逊 Linux 或亚马逊 Linux 2。有关兼容实例类型和亚马逊优化的更多信息 AMIIDs，请参阅亚马逊ECSECS优化。AMIs在本示例中，您启动了一个GPU基于 AMI us-east-1、磁盘大小为 100 GB 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. 启动经过亚马逊ECS优化的Amazon EC2 实例，AMI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。要获取最新的亚马逊ECS优化 AMI ID，请参阅《亚马逊弹性容器服务开发者指南AMIs》中的亚马逊ECS优化 ID。

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

在 Amazon EC2 控制台中，您可以通过响应中的`instance-id`来验证此步骤是否成功。

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

验证 Amazon EC2 实例是否已在集群中注册

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

后续步骤

要了解有关使用亚马逊上的 Deep Learning Containers 进行训练和推理的信息ECS，请参阅亚马逊ECS教程。

训练

> **Note**
>
> 本节介绍如何使用和在 Dee Amazon p Learning Containers for Amazon 弹性容器 PyTorch 服务上运行训练 TensorFlow。

> **Important**
>
> 如果您的账户已经创建了 Amazon ECS 服务相关角色，则除非您在此处指定角色，否则该角色将默认用于您的服务。如果您的任务定义使用 awsvpc 网络模式或将服务配置为使用服务发现，则需要服务相关角色。如果服务使用外部部署控制器、多个目标组或 Elastic Inference 加速器，则也需要该角色，在这种情况下，您不应在此处指定角色。有关更多信息，请参阅《亚马逊ECS开发者指南》ECS中的使用亚马逊服务相关角色。

内容

PyTorch 训练

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。以下示例使用一个示例 Docker 镜像，该镜像将训练脚本添加到 Deep Learning Containers 中。

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。记下输出中的修订版号，并在下一步中使用它。

```
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。您需要上一步中的修订标识符。

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

4. 在 https://console.aws.amazon.com/ecs/v2 中打开控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 任务处于RUNNING状态后，选择任务标识符。
8. 在“日志”下，选择“查看登录信息” CloudWatch。这会将您带到 CloudWatch 控制台以查看训练进度日志。

后续步骤

要在亚马逊上ECS使用 Deep Learning Cont PyTorch ainers 学习推理，请参阅PyTorch 推断。

TensorFlow训练

必须先注册任务定义，然后才能在ECS集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用一个示例 Docker 镜像，该镜像将训练脚本添加到 Deep Learning Containers 中。您可以将此脚本与 TensorFlow 或 TensorFlow 2 配合使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 镜像。

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。记下输出中的修订版号，并在下一步中使用它。

```
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。您需要上一步中的修订版号和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 Amazon ECS 经典游戏机，网址为https://console.aws.amazon.com/ecs/。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster 页面上，选择 Tasks。
7. 任务处于RUNNING状态后，选择任务标识符。
8. 在 “日志” 下，选择 “查看登录信息” CloudWatch。这会将您带到 CloudWatch 控制台以查看训练进度日志。

后续步骤

要在亚马逊上ECS使用 Deep Learning Cont TensorFlow ainers 学习推理，请参阅TensorFlow推断。

推理

本节介绍如何使用 PyTorch和在适用于亚马逊弹性容器服务 (AmazonECS) 的 Dee Amazon p Learning Containers 上运行推理。 TensorFlow

> **Important**
>
> 如果您的账户已经创建了 Amazon ECS 服务相关角色，则除非您在此处指定角色，否则该角色将默认用于您的服务。如果您的任务定义使用 awsvpc 网络模式，则需要服务相关角色。如果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，则也需要该角色，在这种情况下，您不应在此处指定角色。有关更多信息，请参阅《亚马逊ECS开发者指南》ECS中的使用亚马逊服务相关角色。

内容

PyTorch 推断

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。以下示例使用一个示例 Docker 镜像，该镜像向 Deep Learning C CPU on GPU tainers 添加了或推理脚本。

后续步骤

要了解如何在亚马逊上将自定义入口点与 Deep Learning Containers 配合使用ECS，请参阅。自定义入口点

TensorFlow推断

以下示例使用一个示例 Docker 镜像，该镜像通过主机的命令行向 Deep Learn GPU ing Containers 添加CPU或推理脚本。

CPU基于推理

使用以下示例进行CPU基于运行的推理。

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

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mkdir -p /test && cd /test && git clone -b r1.15 https://github.com/
tensorflow/serving.git && tensorflow_model_server --port=8500 --rest_api_port=8501
 --model_name=saved_model_half_plus_two --model_base_path=/test/serving/
tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_cpu"
  ],
  "entryPoint": [
   "sh",
   "-c"
  ],
  "name": "tensorflow-inference-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-cpu-py36-ubuntu18.04",
```

```
  "memory": 8111,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
    "hostPort": 8500,
    "protocol": "tcp",
    "containerPort": 8500
   },
   {
    "hostPort": 8501,
    "protocol": "tcp",
    "containerPort": 8501
   },
   {
    "containerPort": 80,
    "protocol": "tcp"
   }
  ],
  "logConfiguration": {
   "logDriver": "awslogs",
   "options": {
    "awslogs-group": "/ecs/tensorflow-inference-gpu",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "half-plus-two",
    "awslogs-create-group": "true"
   }
  }
 }],
 "volumes": [],
 "networkMode": "bridge",
 "placementConstraints": [],
 "family": "tensorflow-inference"
}
```

2. 注册任务定义。记下输出中的修订版号，并在下一步中使用它。

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

3. 创建亚马逊ECS服务。指定任务定义时，请`revision_id`用上一步输出中任务定义的修订版号替换。

```
aws ecs create-service --cluster ecs-ec2-training-inference \
```

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

4. 通过完成以下步骤来验证服务并获取网络终端节点。

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

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

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

   d. 任务处于RUNNING状态后，选择任务标识符。

   e. 在 “日志” 下，选择 “查看登录信息” CloudWatch。这会将您带到 CloudWatch 控制台查看训练进度日志。

   f. 在 Containers (容器) 下，展开容器详细信息。

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

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，请确保您的公司防火墙没有阻止非标准端口，例如 8501。您可以尝试切换至来宾网络来验证。

GPU基于推理

使用以下示例进行GPU基于运行的推理。

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

```
{
 "requiresCompatibilities": [
  "EC2"
 ],
 "containerDefinitions": [{
  "command": [
   "mkdir -p /test && cd /test && git clone -b r1.15 https://github.com/
tensorflow/serving.git && tensorflow_model_server --port=8500 --rest_api_port=8501
 --model_name=saved_model_half_plus_two --model_base_path=/test/serving/
tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_gpu"
  ],
  "entryPoint": [
   "sh",
   "-c"
  ],
  "name": "tensorflow-inference-container",
  "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference:1.15.0-gpu-py36-cu100-ubuntu18.04",
  "memory": 8111,
  "cpu": 256,
  "resourceRequirements": [{
   "type": "GPU",
   "value": "1"
  }],
  "essential": true,
  "portMappings": [{
    "hostPort": 8500,
    "protocol": "tcp",
    "containerPort": 8500
   },
   {
    "hostPort": 8501,
    "protocol": "tcp",
    "containerPort": 8501
   },
```

```
    {
     "containerPort": 80,
     "protocol": "tcp"
    }
   ],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "/ecs/TFInference",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs",
     "awslogs-create-group": "true"
    }
   }
  }],
  "volumes": [],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "TensorFlowInference"
}
```

2. 注册任务定义。记下输出中的修订版号，并在下一步中使用它。

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

3. 创建亚马逊ECS服务。指定任务定义时，请revision_id用上一步输出中任务定义的修订版号替换。

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

4. 通过完成以下步骤来验证服务并获取网络终端节点。

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

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

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

d. 任务处于RUNNING状态后，选择任务标识符。

e. 在 “日志” 下，选择 “查看登录信息” CloudWatch。这会将您带到 CloudWatch 控制台查看训练进度日志。

f. 在 Containers (容器) 下，展开容器详细信息。

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

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，请确保您的公司防火墙没有阻止非标准端口，例如 8501。您可以尝试切换至来宾网络来验证。

自定义入口点

对于某些图像，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。

修改包含任务定义JSON的文件中的`entryPoint`参数。包括自定义入口点脚本的文件路径。此处显示了一个示例。

```
"entryPoint":[
          "sh",
          "-c",
```

```
            "/usr/local/bin/multi-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"],
```

亚马逊EKS教程

Amazon EKS 教程提供了训练和推理示例，并展示了如何在以下方面设置和使用 Dee Amazon p Learning Containers：

- 亚马逊 Elastic Kubernetes Service（亚马逊）EKS
- Kubeflow 开启 Amazon

Kubeflow on Amazon 是适用于亚马逊 Elastic Kubernetes Service（亚马逊）的 Kubeflow 经过优化的开源发行版。EKS有关更多信息，请参阅 Kubeflow 的Amazon 功能。

> **Note**
>
> 本节中的所有训练和推理示例都在单节点集群上运行。

上的 Kubeflow 安装说明 Amazon 提供了在部署 Kubeflow Amazon 发行版之前创建亚马逊EKS集群的步骤。

内容

- 亚马逊EKS设置
- 自定义入口点
- 在 De Amazon ep Learning Container EKS

亚马逊EKS设置

本节提供安装说明，用于在亚马逊 Elastic Kubernetes Service（亚马逊）上设置运行 Amazon 深度学习容器的深度学习环境。EKS

许可

要使用GPU硬件，请使用具有必要GPU驱动程序的 Amazon 系统映像。我们建议使用经过亚马逊EKS优AMI化的GPU支持版，本指南的后续步骤中将使用该支持。这AMI包括非最终用户许可协议的软

件 Amazon，因此需要最终用户许可协议（EULA）。您必须订阅AMI中的EKS优化版 Amazon Web Services Marketplace 并接受，EULA然后才能在工作节点组AMI中使用。

> **Important**
>
> 要订阅，请访问 Amazon Marketplace。AMI

配置安全设置

要使用 Amazon，EKS您必须拥有一个可以访问多项安全权限的用户账户。这些是用 Amazon Identity and Access Management (IAM) 工具设置的。

1. 按照在IAM您的 Amazon 账户中创建IAM用户中的步骤创建IAM用户或更新现有用户。
2. 获取此用户的凭证。
   a. 打开IAM控制台，网址为https://console.aws.amazon.com/iam/。
   b. 在下方Users，选择您的用户。
   c. 选择 Security Credentials。
   d. 选择 Create access key。
   e. 下载 key pair 或复制信息以备日后使用。
3. 向您的IAM用户添加以下策略。这些政策为亚马逊和亚马逊EKS弹性计算云 (AmazonEC2) 提供了所需的访问权限。IAM
   a. 选择 Permissions。
   b. 选择 Add permissions。
   c. 选择 Create policy。
   d. 从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. 命名策略EKSFullAccess并创建策略。

g. 导航回Grant permissions窗口。

h. 选择 Attach existing policies directly。

i. 搜索EKSFullAccess并选中该复选框。

j. 搜索AWSCloudFormationFullAccess并选中该复选框。

k. 搜索AmazonEC2FullAccess并选中该复选框。

l. 搜索IAMFullAccess并选中该复选框。

m. 搜索AmazonEC2ContainerRegistryReadOnly并选中复选框。

n. 搜索AmazonEKS_CNI_Policy并选中复选框。

o. 搜索AmazonS3FullAccess并选中复选框。

p. 接受更改。

网关节点

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

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，您还可以在同一区域启动网关节点。您可以按照这个 10 分钟的快速教程进行操作，该教程将指导您启动DLAMI以用作网关节点。

1. 登录到您的网关节点。
2. 安装或升级 Amazon CLI。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. eksctl通过运行 Amazon EKS 用户指南安装说明中与您的操作系统对应的命令进行安装。有关的更多信息eksctl，另请参阅 eksctl 文档。
4. kubectl按照安装 kubectl 指南中的步骤进行安装。

> **Note**
>
> 您使用的kubectl版本必须与您的 Amazon EKS 集群控制平面版本相差不到一个次要版本。例如，1.18 kubectl 客户端可与 Kubernetes 1.17、1.18 和 1.19 集群配合使用。

5. 通过运行以下命令安装 aws-iam-authenticator。有关的更多信息 aws-iam-authenticator，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 “安全配置” 部分aws configure为IAM用户运行。您正在复制IAM用户的 Amazon 访问密钥，然后是您在IAM控制台中访问的私有访问 Amazon 密钥，然后将其粘贴到来自aws configure的提示中。

GPU集群

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

   - name是您用来管理集群的工具。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。有关支持的亚马逊EKS版本，请参阅可用的亚马逊 EKS Kubernetes 版本。
   - nodes是您想要在集群中安装的实例数量。在本示例中，我们将从三个节点开始。
   - node-type指的是一个实例类。
   - timeout而且*ssh-access *可以不管。
   - ssh-public-key是您要用来登录工作节点的密钥的名称。要么使用你已经使用的安全密钥，要么创建一个新的安全密钥，但一定要 ssh-public-key用为你使用的区域分配的密钥替换。注意：您只需要提供在 Ama EC2 zon 控制台的 “密钥对” 部分中显示的密钥名称。
   - region是启动集群的 Amazon EC2 区域。如果您计划使用位于特定区域（除了 *<us-east-1>*）我们建议您使用相同的区域。 ssh-public-key必须有权在该地区启动实例。

> **Note**
>
> 本指南的其余部分假设 *<us-east-1>* 作为地区。

2. 对命令进行更改后，运行该命令并等待。单节点群集可能需要几分钟，如果您选择创建大型集群，则可能需要更长的时间。

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

您应该可以看到类似于如下输出的内容：

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

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

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

您应该可以看到类似于如下输出的内容：

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

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

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

5. 验证集群中每个节点上都有GPUs可用的

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

CPU集群

请参阅前一节中关于使用eksctl命令启动GPU集群以及修改node-type为使用CPU实例类型的讨论。

哈瓦那集群

请参阅前面关于使用eksctl命令启动GPU集群的讨论，并修改node-type为使用带有 Habana Gaudi 加速器的实例（例如实例类型）。DL1

测试您的集群

1. 您可以在集群上运行kubectl命令以检查其状态。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。如果您进一步浏览该文件夹，则可以找到 ~/。 kube/eksctl/clusters-它保存了使用 eksctl 创建的集群的 kubeconfig 文件。此文件包含一些您理想情况下不必修改的细节，因为这些工具会为您生成和更新配置，但是在故障排除时最好参考一下。
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，确保找到的默认上下文设置kubectl正确会有所帮助。使用以下命令检查此内容：

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

5. 如果未按预期设置该上下文，请使用以下命令修复此问题：

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

管理您的集群

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

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

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

```
$ kubectl get pods
```

如果您设置了多个集群，但它们尚未安装NVIDIA插件，则可以通过以下方式进行安装：

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

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

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

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

清理

使用完集群后，请将其删除，以免产生额外费用。

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

要仅删除 pod，请运行以下命令：

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

要重置访问集群的密钥，请运行以下命令：

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

要删除nodegroup连接到集群的连接，请运行以下命令：

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

要将nodegroup连接到集群，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

要了解有关使用亚马逊上的 Deep Learning Containers 进行训练和推理的信息EKS，请访问训练或推理。

内容

- 训练
- 推理

训练

使用中的步骤创建集群后亚马逊EKS设置，即可使用它来运行训练作业。对于训练，您可以使用CPUGPU、或分布式GPU示例，具体取决于集群中的节点。以下各节中的主题介绍如何使用 PyTorch 训练示例。

内容

- CPU训练
- GPU训练
- 分布式GPU训练

CPU训练

本节用于CPU基于容器的培训。

内容

- PyTorch CPU训练
- TensorFlow CPU训练
- 后续步骤

PyTorch CPU训练

本教程将指导您在单节点 CPU Pod 上训练 PyTorch 模型。

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

```
apiVersion: 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 文件分配给集群kubectl。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“pytorch-training”的名称位于 pytorch.yaml 文件中。它现在将显示在状态中。如果您正在运行任何其他测试或以前运行过某些测试，则它会出现在此列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods
```

您应看到以下输出：

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

5. 检查日志以查看训练输出。

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

您应该可以看到类似于如下输出的内容：

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

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

6. 查看日志以查看训练进度。您也可以继续选中 “get pods” 以刷新状态。当状态更改为 “Completed” 时，您将知道训练工作已完成。

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什么的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。打开vi或vim并复制并粘贴以下内容。将此文件另存为 `tf.yaml`。你可以将其与 TensorFlow或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 镜像。

```
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 文件分配给集群kubectl。

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

3. 您应看到以下输出：

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

4. 检查状态。任务“tensorflow-training”的名称位于 tf.yaml 文件中。它现在将显示在状态中。如果您正在运行任何其他测试或以前运行过某些测试，则它会出现在此列表中。多次运行此项，直到您看到状态更改为“Running (正在运行)”。

```
$ kubectl get pods
```

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。您也可以继续选中 “get pods” 以刷新状态。当状态更改为 “Completed” 时，您将知道训练工作已完成。

后续步骤

要在亚马逊上EKS使用 Deep Learning Contain TensorFlow ers 进行CPU基于学习的推理，请参阅TensorFlow CPU推断。

后续步骤

要在亚马逊上EKS使用 Deep Learning Contain PyTorch ers 进行CPU基于学习的推理，请参阅PyTorch CPU推断。

GPU训练

本节用于GPU基于集群的培训。

内容

- PyTorch GPU训练
- TensorFlow GPU训练

PyTorch GPU训练

本教程将指导您在单节点GPU集群 PyTorch 上进行训练。

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

```
apiVersion: 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 文件分配给集群kubectl。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“pytorch-training”的名称位于 pytorch.yaml 文件中。它现在将显示在状态中。如果您正在运行任何其他测试或以前运行过某些测试，则它会出现在此列表中。多次运行此命令，直到看到状态更改为 “Running”。

```
$ kubectl get pods
```

您应看到以下输出：

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

5. 检查日志以查看训练输出。

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

您应该可以看到类似于如下输出的内容：

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

6. 查看日志以查看训练进度。您也可以继续选中 “get pods” 以刷新状态。当状态变为 `Completed` “” 时，训练工作就完成了。

后续步骤

要在亚马逊上EKS使用 Deep Learning Contain PyTorch ers 进行GPU基于学习的推理，请参阅PyTorch GPU推断。

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什么的说明。此 pod 文件将下载 Keras 并运行 Keras 示例。此示例使用 TensorFlow 框架。打开vi或vim并复制并粘贴以下内容。将此文件另存为 `tf.yaml`。你可以将其与 TensorFlow或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 镜像。

```
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 文件分配给集群kubectl。

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

3. 您应看到以下输出：

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

4. 检查状态。任务“tensorflow-training”的名称位于 tf.yaml 文件中。它现在将显示在状态中。如果您正在运行任何其他测试或以前运行过某些测试，则它会出现在此列表中。多次运行此命令，直到看到状态更改为 “Running”。

```
$ kubectl get pods
```

您应看到以下输出：

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

5. 检查日志以查看训练输出。

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

您应该可以看到类似于如下输出的内容：

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

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

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

6. 查看日志以查看训练进度。您也可以继续选中 “get pods” 以刷新状态。当状态变为 “Completed” 时，训练作业就完成了。

后续步骤

要在亚马逊上EKS使用 Deep Learning Contain TensorFlow ers 进行GPU基于学习的推理，请参阅TensorFlow GPU推断。

分布式GPU训练

本节用于在多节点GPU集群上运行分布式训练。

内容

- 设置集群以进行分布式训练
- PyTorch分布式GPU训练

设置集群以进行分布式训练

要在上运行分布式训练EKS，你需要在集群上安装以下组件。

- Kubeflow 的默认安装，其中包含所需的组件，例如 PyTorch 运算符和插件。NVIDIA
- MPI运营商。

下载并运行脚本以在集群中安装所需的组件。

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

PyTorch分布式GPU训练

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

1. 确认已安装 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
```

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。将其保存在名为的文件中distributed.yaml。

```
apiVersion: kubeflow.org/v1
kind: PyTorchJob
metadata:
  name: "kubeflow-pytorch-gpu-dist-job"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
pytorch-training:1.7.1-gpu-py36-cu110-ubuntu18.04-example"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
    Worker:
      replicas: 2
      restartPolicy: OnFailure
      template:
        spec:
```

```
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-
pytorch-training:1.7.1-gpu-py36-cu110-ubuntu18.04-example"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
            resources:
              limits:
                nvidia.com/gpu: 1
```

4. 使用您刚刚创建的 pod 文件运行分布式训练作业。

```
$ 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>
```

推理

使用中的步骤创建集群后亚马逊EKS设置，即可使用它来运行推理作业。为了进行推断，您可以根据集群中的节点使用CPU或GPU示例。推理仅支持单节点配置。以下主题介绍如何在使用 PyTorch、和 TensorFlow时EKS使用 Dee Amazon p Learning Containers 运行推理。

内容

- CPU推断
- GPU推断

CPU推断

本节将指导您使用 PyTorch和 TensorFlow在EKSCPU集群的 Deep Learning Containers 上运行推理。

有关深度学习容器的完整列表，请参阅可用的深度学习容器映像。

内容

PyTorch CPU推断

在这种方法中，您可以创建一个 Kubernetes 服务和一个用于运行CPU推理的部署。PyTorchKubernetes 服务公开了一个进程及其端口。在创建 Kubernetes 服务时，您可以指定要使用的服务类型。ServiceTypes默认 ServiceType 为 ClusterIP。部署负责确保一定数量的 pod 始终处于启动和运行状态。

1. 创建命名空间。你可能需要更改 kubeconfig 以指向正确的集群。确认您已设置了 “training-cpu-1” 或将其更改为集群的配置。CPU有关设置集群的更多信息，请参阅亚马逊EKS设置。

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

2. （使用公共模型时的可选步骤。）在可安装的网络位置设置模型，例如在 Amazon S3 中。有关如何将经过训练的模型上传到 S3 的信息，请参阅TensorFlow CPU推断。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、使用的 PyTorch 推理图像以及模型的位置。此示例使用公共模型，因此您无需对其进行修改。

   ```
   ---
   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:
        - multi-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” 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. 因为 serviceType 这里是 ClusterIP，所以你可以将端口从容器转发到你的主机。

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

8. 服务器启动后，您现在可以使用以下命令从不同的窗口运行推理：

```
$ 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
```

有关在使用完集群后对其进行清理的信息，请参阅清EKS理。

TensorFlow CPU推断

在本教程中，您将创建一个 Kubernetes 服务和一个用于运行CPU推理的部署。TensorFlowKubernetes 服务公开了一个进程及其端口。在创建 Kubernetes 服务时，您可以指定要使用的服务类型。ServiceTypes默认 ServiceType 为 ClusterIP。部署负责确保一定数量的 pod 始终处于启动和运行状态。

1. 创建命名空间。你可能需要更改 kubeconfig 以指向正确的集群。确认您已设置了 “training-cpu-1” 或将其更改为集群的配置。CPU有关设置集群的更多信息，请参阅亚马逊EKS设置。

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

2. 可以用不同的方式检索用于推理的模型，例如使用共享卷和 Amazon S3。由于 Kubernetes 服务需要访问亚马逊 S3 和亚马逊ECR，因此您必须将您的 Amazon 证书存储为 Kubernetes 密钥。在本示例中，使用 S3 存储和获取经过训练的模型。

验证您的 Amazon 凭证。他们必须具有 S3 写入权限。

```
$ 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. 在您的主目录中创建一个名为`secret.yaml`的文件，其中包含以下内容。此文件用于存储密钥。

```
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 tensorf low 服务存储库。

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

8. 将预训练saved_model_half_plus_two_cpu模型同步到您的 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。你可以将其与 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 镜像。

```
---
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 是 clusterIP，因此您可以将端口从容器转发到主机。

```
$ 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 字符串放在名为的文件中 `half_plus_two_input.json`

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

15. 在模型上运行推理。

```
$ 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
]
}
```

后续步骤

要了解如何在亚马逊上将自定义入口点与 Deep Learning Containers 配合使用EKS，请参阅。自定义入口点

GPU推断

本节介绍如何在EKSGPU集群的 Deep Learning Containers 上运行推理 PyTorch，以及。 TensorFlow

有关深度学习容器的完整列表，请参阅可用的深度学习容器映像。

内容

- PyTorch GPU推断
- TensorFlow GPU推断
- 后续步骤

PyTorch GPU推断

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。在创建 Kubernetes 服务时，您可以指定要使用的服务类型。ServiceTypes默认 ServiceType 为 ClusterIP。部署负责确保一定数量的 pod 始终处于启动和运行状态。

1. 要进行 GPU-base 推断，请安装适用于 Kubernet NVIDIA es 的设备插件。

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

2. 验证运行 nvidia-device-plugin-daemonset是否正常。

```
$ 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。请参阅使用推理一节中提及的将经过训练的模型上传到 S3 的步骤。 TensorFlow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:
        - multi-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” 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 serviceType 这里是 ClusterIP，因此您可以将端口从容器转发到主机（& 符号在后台运行）。

```
$ 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
```

有关在使用完集群后对其进行清理的信息，请参阅清EKS理。

TensorFlow GPU推断

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。在创建 Kubernetes 服务时，您可以指定要使用的服务类型。ServiceTypes默认 ServiceType 为 ClusterIP。部署负责确保一定数量的 pod 始终处于启动和运行状态。

1. 要进行 GPU-base 推断，请安装适用于 Kubernet NVIDIA es 的设备插件：

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

2. 验证运行 nvidia-device-plugin-daemonset是否正常。

```
$ 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。确认您已设置了 “training-gpu-1” 或将其更改为集群的配置。GPU有关设置集群的更多信息，请参阅亚马逊EKS设置。

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

4. 用于推理的模型可通过不同的方式进行检索，例如，使用共享卷、S3 等。由于该服务需要访问 S3 和ECR，因此您必须将 Amazon 凭证存储为 Kubernetes 密钥。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。你可以将其与 TensorFlow 或 TensorFlow 2 一起使用。要将其与 TensorFlow 2 一起使用，请将 Docker 镜像更改为 TensorFlow 2 镜像。

```
---
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” 状态：

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

14. 重复检查状态步骤，直到看到以下 “RUNNING” 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 serviceType 这里是 ClusterIP，因此您可以将端口从容器转发到主机（& 符号在后台运行）：

```
$ 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
]
```

```
}
```

后续步骤

要了解如何在亚马逊上将自定义入口点与 Deep Learning Containers 配合使用EKS，请参阅。自定义入口点

自定义入口点

对于某些图像， 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/multi-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集群访问存放镜像的ECR存储库的IAM权限已过期，或者你使用的 kubectl 来自与创建集群的用户不同的用户，则会收到以下错误。

```
error: unable to recognize "job.yaml": Unauthorized
```

要解决此问题，您需要刷新令IAM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
```

在 De Amazon ep Learning Container EKS

以下是在亚马逊EKS集群上使用 Dee Amazon p Learning Containers 时，命令行中可能返回的常见错误。每个错误的后面都提供了错误的解决方案。

故障排除

主题

设置错误

在您的亚马逊EKS集群上设置 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：已超过上下文截止日期

```
$ 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，请确认您的账户没有超出容量。您也可以尝试在其他区域创建集群。

- 错误：与服务器 localhost: 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：handle 对象：正在从群集中修补对象：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
```

此错误是由于具有不同授权或凭据的多个用户尝试在同一个集群上启动作业时可能出现并发问题。确认您正在正确的集群上启动作业。

- 错误：无法创建应用程序；目录 “/home/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
```

您可以放心地忽略这一警告。但是，您可能需要在该文件夹中进行其他清理。要简化清理工作，请删除该文件夹。

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 集群时看到此错误消息，请再次运行NVIDIA设备插件安装步骤。通过传入特定的配置文件或将您的活动集群切换到目标集群，验证您的目标群集是否正确。

清理错误

清理 Amazon EKS 集群的资源时，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，请使用`aws configure`或导出 Amazon 环境变量验证是否`~/.kube/config`指向正确的集群，以及 Amazon 凭据是否已正确配置。

- 错误：从起始路径中查找应用程序根目录：：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，请确认您位于 ksonnet 应用程序创建的目录中。`ks init`这是运行所在的文件夹。

- 错误：来自服务器 (NotFound) 的错误：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
```

此错误可能是由于删除上下文后尝试访问资源所致。删除默认上下文也会导致相应的资源也被删除。

发布说明

查看为特定机器学习框架、基础架构和 Amazon 服务构建的 Dee Amazon p Learning Containers 的最新版本说明。

> **Tip**
>
> 有关可用深度学习容器的完整列表以及如何提取这些容器的信息，请参阅可用的深度学习容器映像。

PyTorch 深度学习容器

版本	类型	服务	架构	发行说明
2.4	训练	EC2 ECS EKS	X86	Amazon 适用于 PyTorch 2.4 的 Deep Learning Containers（在 EC2ECS、和上训练EKS）：2024 年 9 月 26 日
2.4	训练	SageMaker	X86	Amazon 适用于 PyTorch 2.4 的 Deep Learning Containers（正在训练 SageMaker）：2024 年 9 月 26 日
2.4	推理	EC2 ECS EKS	X86	Amazon PyTorch 2.4 版 Deep Learning Containers（推

版本	类型	服务	架构	发行说明
				断EC2ECS、和EKS）：2024 年 9 月 26 日
2.4	推理	SageMaker	X86	Amazon PyTorch 2.4 版 Deep Learning Containers（推理开启 SageMaker）：2024 年 10 月 3 日
2.4	推理	EC2 ECS EKS	Arm64	Amazon 适用于 Graviton PyTorch 2.4 的 Deep Learning Containers（推断EC2ECS、和EKS）：2024 年 7 月 11 日
2.4	推理	SageMaker	Arm64	Amazon 适用于 Graviton PyTorch 2.4 的 Deep Learning Containers (SageMaker)：2024 年 7 月 11 日

版本	类型	服务	架构	发行说明
2.3	训练	EC2 ECS EKS	X86	Amazon 适用于 PyTorch 2.3 的 Deep Learning Containers（在 EC2ECS、和上训练EKS）：2024 年 5 月 30 日
2.3	训练	SageMaker	X86	Amazon 适用于 PyTorch 2.3 的 Deep Learning Containers（正在训练 SageMaker）：2024 年 5 月 30 日
2.3	推理	EC2 ECS EKS	X86	Amazon PyTorch 2.3 版 Deep Learning Containers（推断EC2ECS、和 EKS）：2024 年 7 月 11 日
2.3	推理	SageMaker	X86	Amazon PyTorch 2.3 版 Deep Learning Containers（推理开启 SageMaker）：2024 年 7 月 11 日

版本	类型	服务	架构	发行说明
2.3	推理	EC2 ECS EKS	Arm64	Amazon 适用于 Graviton PyTorch 2.3 的 Deep Learning Containers 2.3（EC2ECS、和EKS）：2024 年 7 月 11 日
2.3	推理	SageMaker	Arm64	Amazon 适用于 Graviton 的 Deep Learning Conta PyTorch iners 2.3 (SageMaker)：2024 年 7 月 11 日
2.2	训练	EC2 ECS EKS	X86	Amazon 适用于 PyTorch 2.2 的 Deep Learning Containers（在 EC2ECS、和上训练EKS）：2024 年 3 月 20 日
2.2	训练	SageMaker	X86	Amazon PyTorch 2.2 版 Deep Learning Container s（培训开启 SageMaker）：2024 年 3 月 20 日

版本	类型	服务	架构	发行说明
2.2	推理	EC2 ECS EKS	X86	Amazon PyTorch 2.2 版 Deep Learning Containers（推断EC2ECS、和EKS）：2024 年 3 月 20 日
2.2	推理	SageMaker	X86	Amazon PyTorch 2.2 版 Deep Learning Containers（推理开启SageMaker）：2024 年 3 月 20 日
2.2	推理	EC2 ECS EKS	Arm64	Amazon PyTorch 2.2 版 Deep Learning Containers（推断EC2ECS、和EKS）：2024 年 3 月 20 日
2.2	推理	SageMaker	Arm64	Amazon 适用于 Graviton PyTorch 2.2 的 Deep Learning Containers 2.2（EC2ECS、和EKS）：2024 年 4 月 16 日

TensorFlow 深度学习容器

版本	类型	服务	架构	发行说明
2.16	训练	EC2 ECS EKS	X86	Amazon 适用于 TensorFlow 2.16 的 Deep Learning Containers（在EC2ECS、和上训练EKS）：2024 年 8 月 20 日
2.16	训练	SageMaker	X86	Amazon 适用于 TensorFlow 2.16 的 Deep Learning Containers（培训开启 SageMaker）：2024 年 8 月 20 日
2.16	推理	EC2 ECS EKS	X86	Amazon 适用于 TensorFlow 2.16 的 Deep Learning Containers（推断EC2ECS、和EKS）：2024 年 7 月 19 日
2.16	推理	SageMaker	X86	Amazon 适用于 TensorFlow 2.16 的 Deep Learning

版本	类型	服务	架构	发行说明
				Containers（推断开启 SageMaker）：2024 年 7 月 19 日
2.16	推理	EC2 ECS EKS	Arm64	Amazon 适用于 Graviton TensorFlow 2.16 的 Deep Learning Containers（ECS、EC2 和EKS）：2024 年 7 月 19 日
2.16	推理	SageMaker	Arm64	Amazon 适用于 Graviton TensorFlow 2.16 的 Deep Learning Containers (SageMaker)：2024 年 7 月 19 日

接收有关新更新的通知

每当有新DLC内容发布时，您都可以收到通知。通知SNS使用以下主题在 A mazon 上发布。

```
arn:aws:sns:us-west-2:767397762724:dlc-updates
```

当有新消息发布时，会在此DLC处发布消息。该容器的版本、元数据和区域图像URI将包含在消息中。

可以使用几种不同的方法接收这些消息。我们建议您使用以下方法。

1. 打开 Amazon SNS 控制台。
2. 如有必要，在导航栏中将 Amazon 区域更改为美国西部（俄勒冈）。您必须选择创建您订阅的SNS通知的区域。
3. 在导航窗格中，选择 “订阅”、“创建订阅”。
4. 对于 Create subscription 对话框，执行以下操作：
   a. 对于主题 ARN，复制并粘贴以下 Amazon 资源名称 (ARN)：**`arn:aws:sns:us-west-2:767397762724:dlc-updates`**
   b. 对于协议，请从 [Amazon SQS、 Amazon Lambda、电子邮件、电子邮件-] 中选择一个 JSON
   c. 对于 Endpo int，输入您将用于接收通知的资源的电子邮件地址或 Amazon 资源名称 (ARN)。
   d. 选择创建订阅。
5. 您会收到一封确认电子邮件，主题为 “Amazon 通知-订阅确认”。打开该电子邮件，选择确认订阅来完成订阅。

支持策略

Amazon Dee@@ p Learning Containers (DLCs) 简化了深度学习工作负载的图像配置，并使用最新的框架、硬件、驱动程序、库和操作系统进行了优化。本页详细介绍了的框架支持政策DLCs。

支持的框架

参考以下 Dee Amazon p Learning Containers Framework 支持策略表，查看哪些框架和版本受到积极支持。

请参阅补丁结束，以查看对于由原始框架维护团队主动支持的当前版本， Amazon 可以支持多长时间。框架和版本以单框架DLCs形式提供。

> **Note**
>
> 在框架版本 x.y.z 中，x 表示主要版本，y 表示次要版本，z 表示补丁版本。例如，对于 TensorFlow 2.6.5，主版本为 2，次要版本为 6，补丁版本为 5。

常见问题

- 哪些框架版本会获得安全补丁？
- Amazon 发布新框架版本时会发布哪些镜像？
- 哪些图片有新增 SageMaker/Amazon 功能？
- “支持的框架”表中是如何定义当前版本的？
- 如果我运行的版本不在“支持的框架”表中，该怎么办？
- 是否DLCs支持以前版本的 TensorFlow？
- 如何找到支持的框架版本的最新补丁映像？
- 多长时间发布一次新映像？
- 运行工作负载时，能在我的实例上以替代方式安装补丁吗？
- 如果有新的补丁或更新的框架版本可用，会发生什么呢？
- 是否可在不更改框架版本的情况下更新依赖项？
- 对我的框架版本的主动支持何时结束？
- 对于框架版本不再主动维护的映像，会为其安装补丁吗？

- 如何使用旧框架版本？
- 如何保持框架及其版本 up-to-date的支持变更？
- 是否需要商业许可证才能使用 Anaconda 存储库？

哪些框架版本会获得安全补丁？

如果框架版本在 Dee Amazon p Learning Containers Framework 支持策略表中标有 “支持”，则它将获得安全补丁。

Amazon 发布新框架版本时会发布哪些镜像？

我们会在 TensorFlow 和 PyTorch的新版本发布后DLCs不久发布新版本。这包括框架的主要版本、主要的次要版本和 major-minor-patch版本。当新版本的驱动程序和库可用时，我们也会更新映像。有关映像维护的更多信息，请参阅 对我的框架版本的主动支持何时结束？。

哪些图片有新增 SageMaker/Amazon 功能？

新功能通常在最新版本的 for DLCs PyTorch 和中发布 TensorFlow。有关新增 SageMaker 或 Amazon 功能的详细信息，请参阅特定图像的发行说明。有关可用容器的列表DLCs，请参阅 Dee Amazon p Learning Containers 的发行说明。有关映像维护的更多信息，请参阅 对我的框架版本的主动支持何时结束？。

“支持的框架”表中是如何定义当前版本的？

Dee Amazon p Learning Containers Framework Support Policy 表中的当前版本是 Amazon 指在上发布的最新框架版本 GitHub。每个最新版本都包括中驱动程序、库和相关软件包的更新DLC。有关映像维护的信息，请参阅 对我的框架版本的主动支持何时结束？

如果我运行的版本不在“支持的框架”表中，该怎么办？

如果您运行的版本不在 Dee Amazon p Learning Containers Framework 支持策略表中，则可能没有最新的驱动程序、库和相关包。要获得更多 up-to-date版本，我们建议您使用您选择的最新DLC版本升级到支持的框架之一。有关可用容器的列表DLCs，请参阅 Dee Amazon p Learning Containers 的发行说明。

是否DLCs支持以前版本的 TensorFlow？

不是。 我们支持每个框架最新主要版本的最新补丁版本，如Amazon 深度学习容器框架支持政策表中所述，自首次 GitHub 发布之日起 365 天内发布。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？

如何找到支持的框架版本的最新补丁映像？

要将DLC与最新的框架版本一起使用，请浏览DLC GitHub发布标签以找到您选择URI的示例映像，然后使用它来提取最新的可用的 Docker 映像。您选择的框架版本必须在 Dee Amazon p Learn in g Containers 框架支持策略表中标记为支持。

多长时间发布一次新映像？

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安装了补丁的映像。我们会监控新修补的框架版本（例如 TensorFlow 2.9 到 TensorFlow 2.9.1）和新的次要发行版本（例如 TensorFlow 2.9 到 TensorFlow 2.10），并尽早提供它们。当发布带有新版本的 TensorFlow 现有版本时CUDA，我们会DLC针对该版本发布支持新CUDA版本的新版本。 TensorFlow

运行工作负载时，能在我的实例上以替代方式安装补丁吗？

不是。 的补丁更新不DLC是 “就地” 更新。

您必须删除实例上的现有镜像，并在不终止实例的情况下提取最新的容器镜像。

如果有新的补丁或更新的框架版本可用，会发生什么呢？

请定期查看发布说明页面以获取您的映像。我们鼓励您在新的补丁或更新的框架可用时将框架升级。有关可用容器的列表DLCs，请参阅 Dee Amazon p Learning Containers 的发行说明。

是否可在不更改框架版本的情况下更新依赖项？

我们在不更改框架版本的情况下更新依赖项。但是，如果依赖项更新导致不兼容，我们就会创建不同版本的映像。请务必查看 Dee Amazon p Learning Containers 的发行说明，了解更新的依赖项信息。

对我的框架版本的主动支持何时结束？

DLC图像是不可变的。一旦创建，就不会改变。结束对框架版本的主动支持涉及四个主要原因：

- 框架版本（补丁）升级
- Amazon 安全补丁

- 补丁结束日期（已过期）
- 依赖关系 end-of-support

> **Note**
>
> 由于版本补丁升级和安全补丁的频率很高，我们建议您DLC经常查看发行说明页面，并在进行更改时进行升级。

框架版本（补丁）升级

如果您的DLC工作负载基于 TensorFlow 2.7.0，并且在 2.7.1 版本上 TensorFlow发布 GitHub，则会发布 2.7.1 Amazon 版本的新DLC工作负载。 TensorFlow 2.7. TensorFlow 1 版本的新图像发布后，之前版本为 2.7.0 的图像将更长时间保持活跃。版本DLC为 TensorFlow 2.7.0 的版本不会收到更多补丁。然后， TensorFlow 2.7 的DLC发行说明页面将使用最新信息进行更新。没有为每个次要补丁提供单独的发布说明页面。

由于补丁升级而DLCs创建的新版本标有更新的发行标签。如果更改不向后兼容，则标签将更改主要版本而不是次要版本（例如，v1.0 将更改为 v2.0 而不是 v 1.2）。

Amazon 安全补丁

如果您的工作负载基于 TensorFlow 2.7.0 版本的映像并 Amazon 制作了安全补丁，则会发布适用于 TensorFlow 2.7.0 的新版本。DLC TensorFlow 2.7.0 版图像的先前版本已不再活跃维护。有关更多信息，请参阅运行工作负载时，能在我的实例上以替代方式安装补丁吗？有关查找最新版本的步骤DLC，请参阅 如何找到支持的框架版本的最新补丁映像？

由于补丁升级而DLCs创建的新版本标有更新的发行标签。如果更改不向后兼容，则标签将更改主要版本而不是次要版本（例如，v1.0 将更改为 v2.0 而不是 v 1.2）。

补丁结束日期（已过期）

DLCs在 GitHub 发布日期 365 天后，他们的补丁结束日期。

> **Important**
>
> 当有重大框架更新时，我们会例外处理。例如。如果 TensorFlow 1.15 更新到 TensorFlow 2.0，那么我们将在自 GitHub 发布之日起两年内继续支持最新版本的 TensorFlow 1.15，或者在 Origin 框架维护团队取消支持后的六个月内（以较早的日期为准）。

依赖关系 end-of-support

如果您正在使用 Python 3.6 在 TensorFlow 2.7.0 DLC 映像上运行工作负载，并且该版本的 Python 已标记为 end-of-support，则所有基于 Python 3.6 的DLC图像都将不再被主动维护。同样，如果标记了像 Ubuntu 16.04 这样的操作系统版本 end-of-support，则所有依赖于 Ubuntu 16.04 的DLC映像都将不再被主动维护。

对于框架版本不再主动维护的映像，会为其安装补丁吗？

不会。不再主动维护的图像就不会有新版本。

如何使用旧框架版本？

要在较旧DLC的框架版本中使用，请浏览DLC GitHub发布标签以找到您选择URI的映像，然后使用它来提取 docker 镜像。

如何保持框架及其版本 up-to-date的支持变更？

使用DLC发行说明和可用的 Deep Learning Containers 镜像页面，继续关注 up-to-dateDLC框架和版本。

是否需要商业许可证才能使用 Anaconda 存储库？

Anaconda 转向了针对某些用户的商业许可模式。积极维护DLCs已从Anaconda频道迁移到公开可用的开源版本的Conda（conda-forge）。

> **Warning**
>
> 如果您积极使用 Anaconda 在不再积极维护的软件包中安装和管理您的软件包及其依赖关系 DLC，那么如果您确定这些条款适用于您，则您有责任遵守 Anaconda Reposit ory 中的管理许可。或者，您可以迁移到 Dee Amazon p Learning Containers Framework 支持策略表中DLCs列出的当前支持的软件之一，也可以使用 conda-forge 作为源代码安装软件包。

Amazon 深度学习 Containers 中的安全

云安全 Amazon 是重中之重。作为 Amazon 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 Amazon 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础架构。 Amazon 还为您提供可以安全使用的服务。作为的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于 Deep Learning Containers 的合规计划，请参阅划分的范围内Amazon 服务按合规计划划分的范围内的服务。
- 云端安全-您的责任由您使用的 Amazon 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用 Deep Learning Containers 时如何应用分担责任模型。以下主题向您展示了如何配置 Deep Learning Containers 以实现您的安全和合规目标。您还将学习如何使用其他 Amazon 服务来帮助您监控和保护您的 Deep Learning Containers 资源。

有关更多信息，请参阅亚马逊安全EC2、亚马逊安全ECSEKS、亚马逊安全和亚马逊安全 SageMaker。

主题

- Amazon 深度学习 Containers 中的数据保护
- Amazon 深度学习容器中的身份和访问管理 Deep Learning Containers
- Dee Amazon p Learning Containers 中的监控和使用跟踪
- Amazon 深度学习 Containers 的合规性验证
- Amazon 深度学习 Containers 中的弹性
- Amazon 深度学习 Containers 中的基础设施安全

Amazon 深度学习 Containers 中的数据保护

分担责任模型 Amazon 分适用于Dee Amazon p Learning Containers中的数据保护。如本模型所述 Amazon ，负责保护运行所有内容的全球基础架构 Amazon Web Services 云。您负责维护对托管在此基础架构上的内容的控制。您还负责您所使用的 Amazon Web Services 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私FAQ。

出于数据保护目的，我们建议您保护 Amazon Web Services 账户 凭据并使用 Amazon IAM Identity Center 或 Amazon Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用SSL/TLS与 Amazon 资源通信。我们需要 TLS 1.2，建议使用 TLS 1.3。
- 使用API进行设置和用户活动记录 Amazon CloudTrail。有关使用 CloudTrail 跟踪捕获 Amazon 活动的信息，请参阅《Amazon CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 Amazon 加密解决方案以及其中的所有默认安全控件 Amazon Web Services 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 Amazon 通过命令行界面或访问时需要 FIPS 140-3 经过验证的加密模块API，请使用端点。FIPS有关可用FIPS端点的更多信息，请参阅联邦信息处理标准 (FIPS) 140-3。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括你使用控制台、、API或 Amazon Web Services 服务 使用 Deep Learning Containers 或其他容器时 Amazon SDKs。 Amazon CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您URL向外部服务器提供，我们强烈建议您不要在中包含凭据信息，URL以验证您对该服务器的请求。

Amazon 深度学习容器中的身份和访问管理 Deep Learning Containers

Amazon Identity and Access Management (IAM) Amazon Web Services 服务 可以帮助管理员安全地控制对 Amazon 资源的访问权限。IAM管理员控制谁可以进行身份验证（登录）和授权（拥有权限）来使用 Deep Learning Containers 资源。IAM无需支付额外费用即可使用。 Amazon Web Services 服务

有关身份和访问管理的更多信息，请参阅亚马逊的身份和访问管理EC2、亚马逊的身份和访问管理、亚马逊的身份和访问管理ECSEKS、亚马逊的身份和访问管理以及亚马逊的身份和访问管理SageMaker。

主题

- 使用身份进行身份验证
- 使用策略管理访问
- IAM与亚马逊合作 EMR

使用身份进行身份验证

身份验证是您 Amazon 使用身份凭证登录的方式。您必须以 Amazon Web Services 账户根用户、IAM 用户身份或通过担任IAM角色进行身份验证（登录 Amazon）。

如果您 Amazon 以编程方式访问，则会 Amazon 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 Amazon 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM用户指南》中的API请求Amazon 签名版本 4。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， 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IAM中的Amazon 多重身份验证。

Amazon Web Services 账户 root 用户

创建时 Amazon Web Services 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 Amazon Web Services 服务 和资源。此身份被称为 Amazon Web Services 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以 root 用户身份登录的任务的完整列表，请参阅《用户指南》中的 “需要根用户凭据的IAM任务”。

IAM 用户和群组

IAM用户是您内部 Amazon Web Services 账户 对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时证书，而不是创建拥有密码和访问密钥等长期凭证的IAM用户。但是，如果您有需要IAM用户长期凭证的特定用例，我们建议您轮换访问密钥。有关更多信息，请参阅《IAM用户指南》中的针对需要长期凭证的用例定期轮换访问密钥。

IAM群组是指定IAM用户集合的身份。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，您可以拥有一个名为的群组，IAMAdmins并授予该群组管理IAM资源的权限。

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，请参阅用户指南中的IAMIAM用户用例。

IAM角色

IAM角色是您内部具有特定权限 Amazon Web Services 账户 的身份。它与IAM用户类似，但与特定人员无关。要在中临时扮IAM演角色 Amazon Web Services Management Console，可以从用户切换到

IAM角色（控制台）。您可以通过调用 Amazon CLI 或 Amazon API操作或使用自定义操作来代入角色URL。有关使用角色的方法的更多信息，请参阅《IAM用户指南》中的代入角色的方法。

IAM具有临时证书的角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM用户指南》中的为第三方身份提供商创建角色。
- 临时IAM用户权限-IAM 用户或角色可以代入一个IAM角色，为特定任务临时获得不同的权限。
- 跨账户访问-您可以使用IAM角色允许其他账户中的某人（受信任的委托人）访问您账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 Amazon Web Services 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅IAM用户指南IAM中的跨账户资源访问权限。
- 跨服务访问 — 有些 Amazon Web Services 服务 使用其他 Amazon Web Services 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序EC2或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS)-当您使用IAM用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS使用调用委托人的权限 Amazon Web Services 服务以及 Amazon Web Services 服务 向下游服务发出请求的请求。FAS只有当服务收到需要与其他 Amazon Web Services 服务 或资源交互才能完成的请求时，才会发出请求。在这种情况下，您必须具有执行这两个操作的权限。有关提出FAS请求时的政策详情，请参阅转发访问会话。
  - 服务角色-服务IAM角色是服务代替您执行操作的角色。IAM管理员可以在内部创建、修改和删除服务角色IAM。有关更多信息，请参阅《IAM用户指南》 Amazon Web Services 服务中的创建角色以向委派权限。
  - 服务相关角色-服务相关角色是一种链接到的服务角色。 Amazon Web Services 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 Amazon Web Services 账户 ，并且归服务所有。IAM管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon 上运行的应用程序 EC2 — 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证书。这比在EC2实例中存储访问密钥更可取。要为EC2实例分配 Amazon 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含角色并允许在EC2实例上运行的程序获得临时证书。有关更多信息，请参阅IAM用户指南中的使用IAM角色向在 Amazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 Amazon 通过创建策略并将其附加到 Amazon 身份或资源来控制中的访问权限。策略是其中的一个对象 Amazon ，当与身份或资源关联时，它会定义其权限。 Amazon 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以JSON文档的 Amazon 形式存储在中。有关JSON策略文档结构和内容的更多信息，请参阅《IAM用户指南》中的JSON策略概述。

管理员可以使用 Amazon JSON策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对其所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以将IAM策略添加到角色中，用户可以代入角色。

IAM无论您使用何种方法执行操作，策略都会定义该操作的权限。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 Amazon Web Services Management Console Amazon CLI、或获取角色信息 Amazon API。

基于身份的策略

基于身份的策略是可以附加到身份（例如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用户指南中的使用客户托管策略定义自定义IAM权限。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 Amazon Web Services 账户。托管策略包括 Amazon 托管策略和客户托管策略。要了解如何在托管策略或内联策略之间进行选择，请参阅《IAM用户指南》中的在托管策略和内联策略之间进行选择。

基于资源的策略

基于资源的JSON策略是您附加到资源的策略文档。基于资源的策略的示例包括IAM角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 Amazon Web Services 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略IAM中使用 Amazon 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs与基于资源的策略类似，尽管它们不使用JSON策略文档格式。

Amazon S3 Amazon WAF、和亚马逊VPC就是支持的服务示例ACLs。要了解更多信息ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

Amazon 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界-权限边界是一项高级功能，您可以在其中设置基于身份的策略可以向IAM实体（IAM用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM用户指南》中的IAM实体的权限边界。
- 服务控制策略 (SCPs)-SCPs 是为中的组织或组织单位 (OU) 指定最大权限的JSON策略 Amazon Organizations。 Amazon Organizations 是一项用于对您的企业拥有的多 Amazon Web Services 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。对成员账户中的实体（包括每个实体）的权限进行了SCP限制 Amazon Web Services 账户根用户。有关 Organization SCPs s 和的更多信息，请参阅《Amazon Organizations 用户指南》中的服务控制策略。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 Amazon 确定是否允许请求，请参阅IAM用户指南中的策略评估逻辑。

IAM与亚马逊合作 EMR

您可以 Amazon Identity and Access Management 与 Amazon 一起使用EMR来定义用户、 Amazon 资源、群组、角色和策略。您还可以控制这些用户和角色可以访问哪些 Amazon 服务。

有关在亚马逊IAM上使用的更多信息EMR，请参阅亚马逊Amazon 身份和访问管理EMR。

Dee Amazon p Learning Containers 中的监控和使用跟踪

你的 D Amazon eep Learning Containers 不附带监控工具。有关监控的信息，请参阅监控和优化、GPU监控亚马逊、监控亚马逊EC2、监控亚马逊ECS和监控 A EKS mazon SageMaker Studio。

使用情况跟踪

Amazon 使用客户反馈和使用信息来提高我们向客户提供的服务和软件的质量。我们在支持的 Dee Amazon p Learning Containers 中增加了使用数据收集功能，以便更好地了解客户的使用情况并指导未来的改进。默认情况下，Deep Learning Containers 的使用情况跟踪处于激活状态。客户可以随时更改其设置，以激活或停用使用情况跟踪。

Dee Amazon p Learning Containers 的使用情况跟踪会收集用于容器的实例 ID、框架、框架版本、容器类型和 Python 版本。 Amazon 还会记录它接收此元数据的事件时间。

不会收集或保留有关容器内使用的命令的信息。不会收集或保留有关容器的其他信息。

要选择退出使用情况跟踪，请将OPT_OUT_TRACKING环境变量设置为 true。

```
OPT_OUT_TRACKING=true
```

故障率跟踪

使用第一方 Dee Amazon SageMaker Amazon p Learning Containers 容器时， SageMaker 团队将收集故障率元数据以提高 Amazon 深度学习容器的质量。默认情况下，Dee Amazon p Learning Containers 的故障率跟踪处于活动状态。客户可以在创建 Amazon SageMaker 端点时更改其设置以激活或停用故障率跟踪。

Dee Amazon p Learning Containers 的故障率跟踪会收集实例 ID、ModelServer 名称ErrorType、ModelServer 版本和ErrorCode。 Amazon 还会记录它接收此元数据的事件时间。

不会收集或保留有关容器内使用的命令的信息。不会收集或保留有关容器的其他信息。

要选择退出故障率跟踪，请将OPT_OUT_TRACKING环境变量设置为true。

```
OPT_OUT_TRACKING=true
```

以下框架版本中的使用情况跟踪

虽然我们建议更新到支持的 Deep Learning Containers，但要选择退出使用这些框架的 Deep Learning Containers 的使用情况跟踪，请将`OPT_OUT_TRACKING`环境变量设置为 true，然后使用自定义入口点来禁用对以下服务的调用：

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

Amazon 深度学习 Containers 的合规性验证

作为多个合规计划的一部分，第三方审计师评估服务的安全 Amazon 性和合规性。有关支持的合规计划的信息，请参阅亚马逊合规验证EC2、亚马逊合规验证ECS、亚马逊合规验证和亚马逊EKS合规验证SageMaker。

有关特定合规计划范围内的 Amazon 服务列表，请参阅按合规计划划分的划分的范围内的服务。有关一般信息，请参阅合规计划。

您可以使用下载第三方审计报告 Amazon Artifact。有关更多信息，请参阅在 Artifact 中 tif Amazon act 中下载报告。

您在使用 Deep Learning Containers 时的合规责任取决于您的数据的敏感度、贵公司的合规目标以及适用的法律和法规。 Amazon 提供了以下资源来帮助实现合规性：

- 安全性与合规性快速入门指南安全性与合规性快速入门指南 - 这些部署指南讨论了架构注意事项，并提供了在 Amazon上部署基于安全性和合规性的基准环境的步骤。
- 合规资源 — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- 使用Amazon Config 开发人员指南中的规则评估资源 — 该 Amazon Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- Amazon Security Hub— 此 Amazon 服务可全面了解您的安全状态 Amazon ，帮助您检查是否符合安全行业标准和最佳实践。

Amazon 深度学习 Containers 中的弹性

Amazon 全球基础设施是围绕 Amazon 区域和可用区构建的。 Amazon 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作

在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅Amazon 全球基础设施。

有关有助于支持您的数据弹性和备份需求的功能的信息，请参阅 Amazon 的弹性EC2、Amaz on 的弹性和亚马逊EKS SageMaker的弹性。

Amazon 深度学习 Containers 中的基础设施安全

Dee Amazon p Learning Containers 的基础设施安全由亚马逊EC2ECS、亚马逊EKS、亚马逊或提供支持 SageMaker。有关更多信息，请参阅亚马逊的基础设施安全EC2、亚马逊的基础设施安全ECSEKS、亚马逊的基础设施安全和亚马逊的基础设施安全 SageMaker。

Deep Learning Containers 文档历史记录开发者指南

下表描述了此版本的 Deep Learning Containers 的文档。

- API版本：最新
- 最新文档更新：2020 年 2 月 26 日

变更	说明	日期
深度学习容器开发者指南发布	开发者指南中添加了 Deep Learning Containers 设置和教程。	2020 年 2 月 17 日

Amazon 术语表

有关最新的 Amazon 术语，请参阅《Amazon Web Services 词汇表参考》中的 Amazon 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。