AWS 深度学习容器

开发人员指南

AWS 深度学习容器: 开发人员指南

Table of Contents

什么是 AWS Deep Learning Containers？

欢迎阅读 AWS Deep Learning Containers 的用户指南。

AWS Deep Learning Containers (Deep Learning Containers) 是一组 Docker 映像，用于在 TensorFlow、TensorFlow 2、PyTorch 和 MXNet 中训练和处理模型。Deep Learning Containers 为优化环境提供了 TensorFlow、MXNet、Nvidia CUDA（适用于 GPU 实例）以及 Intel MKL（适用于 CPU 实例）库且在 Amazon Elastic Container Registry(Amazon ECR) 中可用。

关于本指南

本指南可帮助您设置和使用 AWS Deep Learning Containers。本指南还介绍使用 Amazon EC2、Amazon ECS、Deep Learning Containers 和 SageMaker 设置 Amazon EKS 的过程。它介绍了用于训练和推导的深度学习的几种常见使用案例。本指南还为每个框架提供了几个教程。

Python 2 支持

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 Deep Learning Containers 版本将继续可用。但是，只有在开源社区针对这些版本发布了安全修补程序时，我们才会为 Python 2 Deep Learning Containers 提供更新。具有下一个版本 TensorFlow 框架和 PyTorch 框架的 Deep Learning Containers 版本不包含 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，本指南将向您展示如何激活每一个框架并找到相应的教程以便开始使用。

设置 Deep Learning Containers

AWS Deep Learning Containers 的设置过程取决于使用它们的基础设施。以下各节提供有关使用 Amazon EC2、Amazon ECS 和 Amazon EKS 设置 Deep Learning Containers 的信息。有关 Deep Learning Containers 与 SageMaker 结合使用的信息，请参阅通过 SageMaker 文档使用您自己的算法或模型。

主题

Amazon EC2 设置

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

请完成以下步骤来配置您的实例：

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

  有关创建或编辑 IAM 用户的更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 身份权限。
- 启动 Amazon Elastic Compute Cloud 实例（CPU 或 GPU），最好是深度学习基础 AMI。其他 AMI 可运行，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 Amazon EC2 用户指南 中的排查实例的连接问题。
- 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 --no-include-email --region us-east-1 --registry-ids 763104351884)
```

  在上一个命令的开头处包含“$”。

有关 AWS Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> Note
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

后续步骤

要了解有关在 Amazon EC2 上使用 Deep Learning Containers 进行训练和推论，请参阅Amazon EC2 教程 (p. 10)。

Amazon ECS 设置

本部分介绍如何使用 Amazon Elastic Container Service 设置 AWS Deep Learning Containers。

目录

先决条件

本设置指南假设您已完成以下先决条件：

- 安装并配置最新版本的 AWS CLI。有关安装或升级 AWS CLI 的更多信息，请参阅安装 AWS Command Line Interface。
- 完成使用 Amazon ECS 进行设置中的步骤。
- 结果为以下三种情况之一：
  - 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 AWS 服务委派权限。
  - 一个拥有管理员权限的用户已手动创建这些 IAM 角色，使之对所用的账户可用。有关更多信息，请参阅 Amazon Elastic Container Service Developer Guide 中的 Amazon ECS 服务计划程序 IAM 角色和 Amazon ECS 容器实例 IAM 角色。
- Amazon CloudWatch Logs IAM 策略已经添加到 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 Amazon Elastic Container Service Developer Guide 中的 CloudWatch Logs IAM 策略。
- 生成密钥对。有关更多信息，请参阅 Amazon EC2 密钥对。
- 创建新的安全组或更新现有安全组，以便为所需推理服务器打开端口。
  - 对于 MXNet 推理，对于 TCP 流量打开端口 80 和 8081。
  - 对于 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。

针对 Deep Learning Containers 设置 Amazon ECS

本部分介绍如何设置 Amazon ECS 以便使用 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。如果您的任务定义使用 awsvpc 网络模式，或者将服务配置为使用以下任何功能，则需要服务相关角色：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，则不应在此指定角色。有关更多信息，请参阅 Amazon ECS 开发人员指南 中的为 Amazon ECS 使用服务相关角色。

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 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Amazon ECS 开发人员指南 中的在 Amazon ECS 上使用 GPU，以指导您选择实例类型。选择实例类型后，再选择适合您的使用案例的经 ECS 优化的 AMI。对于基于 CPU 的工作，您可以使用经 Amazon Linux 或 Amazon Linux 2 ECS 优化的 AMI。对于基于 GPU 的工作，必须使用 ECS GPU 优化的 AMI 和 p2/p3 实例类型。您可以在 经 Amazon ECS 优化的 AMI 中找到 Amazon ECS-optimized AMI ID。在本示例中，您将在 us-east-1 中启动一个具有基于 GPU 的 AMI 以及 100 GB 磁盘大小的实例。

   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Amazon ECS-optimized AMI 启动 Amazon EC2 实例并将其附加到集群。使用安全组 ID 和您创建的密钥对名称并在以下命令中替换它们。要获取最新的 Amazon ECS-optimized AMI ID，请参阅 Amazon Elastic Container Service Developer Guide 中的 Amazon ECS 优化的 AMI。

```
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，您可以根据响应中的 `instance-id` 验证此步骤是否成功。

现在，您有一个正在运行容器实例的 Amazon ECS 集群。使用以下步骤验证 Amazon EC2 实例已注册到集群。

验证 Amazon EC2 实例是否已注册到集群

1. 在 https://console.amazonaws.cn/ecs/ 上打开 Amazon ECS 控制台。
2. 选择带有注册了 Amazon EC2 实例的集群。
3. 在 Cluster (集群) 页面上，选择 ECS Instances (ECS 实例)。
4. 验证对于在上一步骤中创建的 `instance-id`，Agent Connected (代理已连接) 值是否为 True (真)。此外，记下控制台上显示的可用 CPU 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

要了解有关在 Amazon ECS 上使用 Deep Learning Containers 进行训练和推论，请参阅Amazon ECS 教程 (p. 16)。

Amazon EKS 设置

本指南说明如何使用 Amazon Elastic Kubernetes Service (Amazon EKS) 和 AWS Deep Learning Containers 设置深度学习环境。使用 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 Amazon EKS 文档介绍如何使用 Kubernetes 工具系列。本指南假定您已熟悉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（构成一个集群）。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。您可能还需要根据深度学习神经网络和模型的需求，为您的集群提供不同的实例类型。

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。提供了使用自定义映像的示例。您可以尝试它们来入门，而无需自行创建。

- 构建 AWS Deep Learning Containers 自定义映像 (p. 78)

许可

要使用 GPU 硬件，请使用具有所需 GPU 驱动程序的 Amazon 系统映像。我们建议结合使用经 Amazon EKS 优化的 AMI 与 GPU 支持，本指南的后续步骤中将使用它们。此 AMI 包括需要最终用户许可协议 (EULA) 的非 AWS 软件。您必须在 AWS Marketplace 中订阅经 EKS 优化的 AMI 并接受 EULA，然后才能在工作线程节点组中使用 AMI。

> **Important**
>
> 要订阅该 AMI，请访问 AWS Marketplace。

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。这些是使用 AWS Identity and Access Management (IAM) 工具设置的。

1. 创建 IAM 用户或更新现有 IAM 用户。要了解有关创建或编辑 IAM 用户的更多信息，请参阅 IAM 文档。
2. 获取此用户的凭证。在“Users (用户)”下，选择您的用户，选择“Security Credentials (安全凭证)”，选择“Create access key pair (创建访问密钥对)”，然后下载该密钥对或复制相关信息以供稍后使用。
3. 向此 IAM 用户添加策略。它们提供 Amazon EKS、IAM 和 Amazon Elastic Compute Cloud (Amazon EC2) 所需的访问权限。
4. 搜索 AmazonEKSAdminPolicy，然后选中该复选框。
5. 搜索 AmazonCloudFormationPolicy，然后选中该复选框。
6. 搜索 AmazonEC2FullAccess，然后选中该复选框。

7. 搜索 IAMFullAccess，然后选中该复选框。
8. 搜索 AmazonEC2ContainerRegistryReadOnly，然后选中该复选框。
9. 搜索 AmazonEKS_CNI_Policy，然后选中该复选框。
10. 搜索 AmazonS3FullAccess，然后选中该复选框。
11. 接受更改。

网关节点

要设置 Amazon EKS 集群，请使用开源工具 `eksctl`。我们建议您使用具有深度学习基础 AMI (Ubuntu) 的 Amazon EC2 实例来分配和控制您的集群。您可以在您的计算机或已在运行的 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具有 Ubuntu 16.04 的深度学习基础 AMI (DLAMI)。我们将此称为您的网关节点。

> **Note**
>
> 您的 `eksctl` 版本必须为 1.12 或更高版本。

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，您可以在同一区域中启动网关节点。您可以按照此快速 10 分钟教程操作，该教程将指导您启动 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 AWS 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. 通过运行以下命令安装 eksctl。有关 eksctl 的更多信息，请参阅 AWS 开源博客上的 eksctl 博客文章。

```
$ 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. 通过运行以下命令安装 `kubectl`。有关 kubectl 的详细信息，请参阅安装 kubectl。

```
$ curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/
linux/amd64/kubectl
$ chmod +x ./kubectl
$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. 通过运行以下命令安装 `aws-iam-authenticator`。有关 aws-iam-authenticator 的更多信息，请参阅 Amazon EKS 入门。

```
$ curl -o aws-iam-authenticator https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/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`。您要复制在 IAM 控制台中访问的 IAM 用户的 AWS 访问密钥和 AWS 秘密访问密钥并将这些密钥粘贴到 `aws configure` 中的提示符中。
7. 安装 ksonnet：

```
$ export KS_VER=0.13.1
```

```
$ export KS_PKG=ks_${KS_VER}_linux_amd64
$ wget -O /tmp/${KS_PKG}.tar.gz https://github.com/ksonnet/ksonnet/releases/download/v
${KS_VER}/${KS_PKG}.tar.gz
$ mkdir -p ${HOME}/bin
$ tar -xvf /tmp/$KS_PKG.tar.gz -C ${HOME}/bin
$ sudo mv ${HOME}/bin/$KS_PKG/ks /usr/local/bin
```

GPU 集群

1. 检查以下使用 p3.8xlarge 实例类型创建集群的命令。您将需要修改它，然后再运行。

   - `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。
   - `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。
   - 可以不考虑 `timeout` 和 `*ssh-access *`。
   - `ssh-public-key` 是您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：您只需提供在 Amazon EC2 控制台的“密钥对”部分中看到的密钥名称。
   - `region` 是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 *<us-east-1>*）中的训练数据，我们建议您使用同一区域。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>*。
   - auto-kubeconfig 可以自行运行。
2. 在对该命令进行更改后，即可运行它，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 1.11 \
                         --nodes 3 \
                         --node-type=<p3.8xlarge> \
                         --timeout=40m \
                         --ssh-access \
                         --ssh-public-key <key_pair_name> \
                         --region <us-east-1> \
                         --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，请确保运行以下步骤来安装适用于 Kubernetes 的 NVIDIA 设备插件：

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

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

请参阅上一部分中有关使用 eksctl 命令启动 GPU 集群的讨论，并将 `node-type` 修改为使用 CPU 实例类型。

测试您的集群

1. 您可以在集群上运行 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应修改的一些详细信息，因为工具正在为您生成和更新配置，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，它有助于确保正确地设置由 kubectl 找到的默认上下文。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“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该集群调用 get pods 命令，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 apply -f https://
raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v1.12/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，您将经常在活动的集群之间切换。这样，您就可以协调安排训练或推理，或使用在不同集群上运行的不同框架。

清除

当用完集群后，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
```

要删除附加到集群的 nodegroup，请运行以下命令：

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

要将 nodegroup 附加到集群，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

要了解有关在 Amazon EKS 上使用 Deep Learning Containers 进行训练和推论，请参阅Amazon EKS 教程 (p. 34)。

Deep Learning Containers 入门

以下各节介绍如何使用 Deep Learning Containers 从 AWS 基础设施上的每个框架运行示例代码。有关 Deep Learning Containers 与 SageMaker 结合使用的信息，请参阅通过 SageMaker 文档使用您自己的算法或模型。

主题

Amazon EC2 教程

本部分说明如何使用 MXNet、PyTorch 和 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，您应该已经完成了Amazon EC2 设置 (p. 2)中的步骤。

目录

训练

本部分指导您如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在适用于 Amazon EC2 的 AWS Deep Learning Containers 上运行训练。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> **Note**
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

目录

TensorFlow 训练

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

- 对于基于 CPU 的训练，请运行以下操作。

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

- 对于基于 GPU 的训练，请运行以下操作。

```
$ 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

要了解有关将 TensorFlow 与 Deep Learning Containers 结合使用在 Amazon EC2 上进行推理，请参阅TensorFlow 推理 (p. 13)。

MXNet 训练

要开始从您的 Amazon EC2 利用 MXNet 进行训练，请首先运行以下命令来运行容器：

- 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 v1.4.1 https://github.com/apache/incubator-mxnet.git
python incubator-mxnet/example/image-classification/train_mnist.py
```

- 对于 GPU

```
$ git clone -b v1.4.1 https://github.com/apache/incubator-mxnet.git
python incubator-mxnet/example/image-classification/train_mnist.py --gpus 0
```

后续步骤

要了解有关将 MXNet 与 Deep Learning Containers 结合使用在 Amazon EC2 上进行推理，请参阅MXNet 推理 (p. 15)。

PyTorch 训练

要开始从 Amazon EC2 利用 PyTorch 进行训练，请使用以下命令来运行容器。您必须将 **nvidia-docker** 用于 GPU 映像。

- 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 版本为 19.03 或更高，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
```

后续步骤

要了解有关将 PyTorch 与 Deep Learning Containers 结合使用在 Amazon EC2 上进行推理，请参阅PyTorch 推理 (p. 15)。

推理

本部分指导您如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在适用于 Amazon Elastic Compute Cloud 的 AWS Deep Learning Containers 上运行推理。您也可以使用 Elastic Inference 通过 AWS Deep Learning Containers 运行推理。有关教程以及 Elastic Inference 的更多信息，请参阅在 Amazon EC2 上将 AWS Deep Learning Containers 与 Elastic Inference 配合使用。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> **Note**
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

目录

- PyTorch 推理 (p. 15)

TensorFlow 推理

为了演示如何将 Deep Learning Containers 用于推理，此示例将一个简单的 half plus two 模型用于 TensorFlow Serving。我们建议将深度学习基础 AMI 用于 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
```

使用此处的命令针对此模型通过 Deep Learning Containers 启动 TensorFlow Serving。与用于训练的 Deep Learning Containers 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/sad_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
```

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

如果要调试容器的输出，可以使用容器名称附加到它，如此命令中所示。

```
$ docker attach <your docker container name>
```

在本示例中，您使用了 tensorflow-inference。

TensorFlow 2 推理

为了演示如何将 Deep Learning Containers 用于推理，此示例将一个简单的 half plus two 模型用于 TensorFlow 2 Serving。我们建议将深度学习基础 AMI 用于 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
```

使用此处的命令针对此模型通过 Deep Learning Containers 启动 TensorFlow Serving。与用于训练的 Deep Learning Containers 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/sad_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到输出：

```
$ docker attach <your docker container name>
```

此示例使用了 tensorflow-inference。

MXNet 推理

为开始使用 MXNet 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，请运行以下命令。

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

对于 GPU 实例，请运行以下命令。

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

配置文件包含在容器中。

在您的服务器启动后，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
```

PyTorch 推理

为了开始使用 PyTorch 进行推理，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，以及这篇有关使用 MXNet Model Server 部署 PyTorch 推理的博客。

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 版本为 19.03 或更高，则启动 Docker 时可以使用 --gpus 标志。

配置文件包含在容器中。

在您的服务器启动后，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

本部分说明如何使用 MXNet、PyTorch 和 TensorFlow 在 Amazon ECS 的 AWS Deep Learning Containers 上运行训练和推理。

在开始以下教程之前，您应该已经完成了Amazon ECS 设置 (p. 2)中的步骤。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> Note
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

目录

训练

本部分指导您如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在适用于 Amazon Elastic Container Service 的 AWS Deep Learning Containers 上运行训练。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> **Note**
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)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，或者如果服务配置为使用服务发现，则需要使用服务相关角色。如果服务使用外部部署控制器、多个目标组或 Elastic Inference 加速器（在这种情况下，不应在此处指定角色），则也需要使用此角色。有关更多信息，请参阅 Amazon ECS 开发人员指南 中的为 Amazon ECS 使用服务相关角色。

目录

TensorFlow 训练

您必须先注册任务定义才能在 ECS 集群上运行任务。任务定义是分组在一起的一系列容器。以下示例使用将训练脚本添加到 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-training:1.15.0-
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:1.15.0-gpu-py36-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，以供下一步中使用。

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

3. 使用任务定义创建任务。您需要使用上一步中的修订号。

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

4. 在 https://console.amazonaws.cn/ecs/ 上打开 Amazon ECS 控制台。
5. 选择 ecs-ec2-training-inference 集群。
6. 在 Cluster (集群) 页面上，选择 Tasks (任务)。
7. 当您的任务处于 RUNNING 状态后，请选择任务标识符。
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

要了解有关将 MXNet 与 Deep Learning Containers 结合使用在 Amazon ECS 上进行推理，请参阅MXNet 推理 (p. 27)。

MXNet 训练

您必须先注册任务定义，然后才能在 Amazon Elastic Container Service 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，以供下一步中使用。

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

3. 使用任务定义创建任务。您需要使用上一步中的修订号。

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

4. 在 https://console.amazonaws.cn/ecs/ 上打开 Amazon ECS 控制台。
5. 选择 `ecs-ec2-training-inference` 集群。
6. 在 Cluster (集群) 页面上，选择 Tasks (任务)。
7. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。
8. 在 Containers (容器) 下，展开容器详细信息。
9. 在 Log Configuration (日志配置) 中，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

要了解有关将 TensorFlow 与 Deep Learning Containers 结合使用在 Amazon ECS 上进行推理，请参阅TensorFlow 推理 (p. 24)。

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:1.3.1-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:1.3.1-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，以供下一步中使用。

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

3. 使用任务定义创建任务。您需要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. 在 https://console.amazonaws.cn/ecs/ 上打开 Amazon ECS 控制台。
5. 选择 `ecs-ec2-training-inference` 集群。
6. 在 Cluster (集群) 页面上，选择 Tasks (任务)。
7. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。
8. 在 Containers (容器) 下，展开容器详细信息。
9. 在 Log Configuration (日志配置) 中，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

要了解有关将 PyTorch 与 Deep Learning Containers 结合使用在 Amazon ECS 上进行推理，请参阅PyTorch 推理 (p. 31)。

推理

本部分指导您如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在适用于 Amazon Elastic Container Service (Amazon ECS) 的 AWS Deep Learning Containers 上运行推理。您也可以使用 Elastic Inference 通过 AWS Deep Learning Containers 运行推理。有关教程以及 Elastic Inference 的更多信息，请参阅在 Amazon ECS 上将 AWS Deep Learning Containers 与 Elastic Inference 配合使用。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。

> **Note**
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)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，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（在这种情况下，不应在此处指定角色），则也需要使用此角色。有关更多信息，请参阅 Amazon ECS 开发人员指南中的为 Amazon ECS 使用服务相关角色。

目录

TensorFlow 推理

以下示例使用通过主机的命令行将 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，以供下一步中使用。

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

3. 创建 Amazon ECS 服务。当您指定任务定义时，请将 `revision_id` 替换为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. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8501 的 IP 地址以供下一步使用。

   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**
>
> If you are unable to connect to the external IP address, be sure that your corporate firewall is not blocking non-standards ports, like 8501. You can try switching to a guest network to verify.

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，以供下一步中使用。

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

3. 创建 Amazon ECS 服务。当您指定任务定义时，请将 `revision_id` 替换为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. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8501 的 IP 地址以供下一步使用。

   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

   If you are unable to connect to the external IP address, be sure that your corporate firewall is not blocking non-standards ports, like 8501. You can try switching to a guest network to verify.

MXNet 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。以下示例使用通过主机的命令行将 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 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，以供下一步中使用。

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

3. 创建 Amazon ECS 服务。当您指定任务定义时，请将 `revision_id` 替换为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. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 当您的任务处于 RUNNING 状态后，请选择任务标识符。
e. 在 Containers (容器) 下，展开容器详细信息。
f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8081 的 IP 地址以供下一步使用。
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 external IP 地址替换为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，以供下一步中使用。

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

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

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

3. 验证服务并获取终端节点。

   a. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8081 的 IP 地址以供下一步使用。

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。将 `external IP` 地址替换为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，以供下一步中使用。

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

3. 创建 Amazon ECS 服务。当您指定任务定义时，请将 `revision_id` 替换为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. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 当您的任务处于 `RUNNING` 状态后，请选择任务标识符。

   e. 在 Containers (容器) 下，展开容器详细信息。

   f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8081 的 IP 地址以供下一步使用。

   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 `external IP` 地址替换为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，以供下一步中使用。

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

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

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

3. 通过完成以下步骤来验证服务并获取网络终端节点。

   a. 在 https://console.amazonaws.cn/ecs/ 上打开 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. 一旦您的任务处于 RUNNING 状态，请选择任务标识符。
   e. 在 Containers (容器) 下，展开容器详细信息。
   f. 在 Name (名称) 和 Network Bindings (网络绑定) 下，记下 External Link (外部链接) 下的端口 8081 的 IP 地址以供下一步使用。
   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。将 external IP 地址替换为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。

修改包含您的任务定义的 JSON 文件中的 entryPoint 参数。将文件路径包含到自定义入口点脚本中。这里显示了一个例子。

```
"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

本部分说明如何使用 MXNet、PyTorch 和 TensorFlow 在 EKS 的 AWS Deep Learning Containers 上运行训练和推理。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，您应该已经完成了Amazon EKS 设置 (p. 4)中的步骤。

目录

训练

在运行集群后，即可尝试训练任务。本部分将指导您完成 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 训练示例。

目录

CPU 训练

本部分介绍基于 CPU 的容器上的训练。

有关 Deep Learning Containers 的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。有关最佳配置设置（如果您使用 Intel 数学内核库 (MKL)）的提示，请参阅AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)。

目录

MXNet CPU 训练

本教程指导您在单节点 CPU 集群上使用 MXNet 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 或 vim，并复制和粘贴以下内容。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 kubectl 将 pod 文件分配到集群。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“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。当状态变为“`Completed`”时，训练工作就会完成。

后续步骤

要了解有关将 MXNet 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 CPU 的推理，请参阅MXNet CPU 推理 (p. 53)。

TensorFlow CPU 训练

本教程指导您在单节点 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 或 vim，并复制和粘贴以下内容。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:1.15.0-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 kubectl 将 pod 文件分配到集群。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“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。当状态变为 `Completed` 时，您将知道该训练任务已完成。

后续步骤

要了解有关将 TensorFlow 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 CPU 的推理，请参阅TensorFlow CPU 推理 (p. 55)。

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 或 vim，并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:1.3.1-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 kubectl 将 pod 文件分配到集群。

```
$ 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。当状态变为 `Completed` 时，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。

后续步骤

要了解有关将 PyTorch 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 CPU 的推理，请参阅PyTorch CPU 推理 (p. 58)。

GPU 训练

本部分介绍在基于 GPU 的集群上进行训练。

有关 Deep Learning Containers的完整列表，请参阅Deep Learning Containers 映像 (p. 72)。有关最佳配置设置（如果您使用 Intel 数学内核库 (MKL)）的提示，请参阅AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)。

目录

MXNet GPU 训练

本教程指导您在单节点 GPU 集群上使用 MXNet 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 或 vim，并复制和粘贴以下内容。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 kubectl 将 pod 文件分配到集群。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“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。当状态变为“Completed”时，训练工作就会完成。

后续步骤

要了解有关将 MXNet 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅MXNet GPU 推理 (p. 60)。

TensorFlow GPU 训练

本教程指导您在单节点 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 或 vim，并复制和粘贴以下内容。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:1.15.0-gpu-py36-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 kubectl 将 pod 文件分配到集群。

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

3. 您应看到以下输出：

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

4. 检查状态。作业“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。当状态变为“Completed”时，训练工作就会完成。

后续步骤

要了解有关将 TensorFlow 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅TensorFlow GPU 推理 (p. 62)。

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 或 vim，并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:1.3.1-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 kubectl 将 pod 文件分配到集群。

```
$ 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。当状态变为“Completed”时，训练工作就会完成。

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

要了解有关将 PyTorch 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅PyTorch GPU 推理 (p. 66)。

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. 72)。

目录

在 EKS 上运行分布式训练将使用 Kubeflow。Kubeflow 项目致力于简单、便携且可扩展地在 Kubernetes 上部署机器学习 (ML) 工作流。

通过安装 Kubeflow 设置您的集群进行分布式训练

1. 安装 Kubeflow。

```
$ export KUBEFLOW_VERSION=0.4.1
$ curl https://raw.githubusercontent.com/kubeflow/kubeflow/v${KUBEFLOW_VERSION}/
scripts/download.sh | bash
```

2. 使用 Kubeflow 程序包时，您将很快遇到 Github API 限制。您需要创建一个 Github 令牌并按以下方式导出它。您无需选择任何范围。

```
$ export GITHUB_TOKEN=<token>
```

MXNet 分布式 GPU 训练

本教程将指导您在多节点 GPU 集群上使用 MXNet 进行分布式训练。它使用 Parameter Server。要在 EKS 上运行 MXNet 分布式训练，我们将使用名为 `MXJob` 的 Kubernetes MXNet-operator。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）。

1. 设置命名空间。

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

2. 设置应用程序名称并将其初始化。

```
$ APP_NAME=kubeflow-mx-ps; ks init ${APP_NAME}; cd ${APP_NAME}
```

3. 将默认环境使用的命名空间更改为 `${NAMESPACE}`。

```
$ ks env set default --namespace ${NAMESPACE}
```

4. 安装用于 kubeflow 的 MXNet-operator。这是使用参数服务器运行 MXNet 分布式训练所需的。

```
$ ks registry add kubeflow github.com/kubeflow/kubeflow/tree/${KUBEFLOW_VERSION}/
kubeflow
$ ks pkg install kubeflow/mxnet-job@v${KUBEFLOW_VERSION}
```

5. 生成与 Kubernetes 兼容的 jsonnet 组件清单文件。

```
$ ks generate mxnet-operator mxnet-operator
```

6. 应用配置设置。

```
$ ks apply default -c mxnet-operator
```

7. 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

您的第一个任务是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 个任务模式：计划程序、服务器和工作线程。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. 根据之前的讨论，您将根据您的要求修改以下代码块并将其保存在名为 `mx_job_dist.yaml` 的文件中。

```
apiVersion: "kubeflow.org/v1alpha1"
kind: "MXJob"
metadata:
  name: "gpu-dist-job"
spec:
  jobMode: "dist"
  replicaSpecs:
    - replicas: 1
      mxReplicaType: SCHEDULER
      PsRootPort: 9000
      template:
        spec:
          containers:
            - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.4.1-gpu-py36-cu100-ubuntu16.04-example
```

```
            name: mxnet
          restartPolicy: OnFailure
    - replicas: 2
      mxReplicaType: SERVER
      template:
        spec:
          containers:
            - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.4.1-gpu-py36-cu100-ubuntu16.04-example
              name: mxnet
          restartPolicy: OnFailure
    - replicas: 2
      mxReplicaType: WORKER
      template:
        spec:
          containers:
            - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.4.1-gpu-py36-cu100-ubuntu16.04-example
              name: mxnet
              command: ["python"]
              args: ["/incubator-mxnet/example/image-classification/train_mnist.py","--
num-epochs","1","--num-layers","2","--kv-store","dist_device_sync","--gpus","0,1"]
              resources:
                limits:
                  nvidia.com/gpu: 2
          restartPolicy: OnFailure
```

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=gpu-dist-job
kubectl get -o yaml mxjobs $JOB
```

该输出值应该类似于以下内容：

```
apiVersion: kubeflow.org/v1alpha1
kind: MXJob
metadata:
  creationTimestamp: 2019-03-21T22:00:38Z
  generation: 1
  name: gpu-dist-job
  namespace: default
  resourceVersion: "2523104"
  selfLink: /apis/kubeflow.org/v1alpha1/namespaces/default/mxjobs/gpu-dist-job
  uid: c2e67f05-4c24-11e9-a6d4-125f5bb10ada
spec:
  RuntimeId: j1ht
  jobMode: dist
  mxImage: jzp1025/mxnet:test
  replicaSpecs:
  - PsRootPort: 9000
    mxReplicaType: SCHEDULER
    replicas: 1
    template:
```

```
      metadata:
        creationTimestamp: null
      spec:
        containers:
        - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.4.1-gpu-py36-cu100-ubuntu16.04-example
          name: mxnet
          resources: {}
        restartPolicy: OnFailure
  - PsRootPort: 9091
    mxReplicaType: SERVER
    replicas: 2
    template:
      metadata:
        creationTimestamp: null
      spec:
        containers:
        - image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-
training:1.4.1-gpu-py36-cu100-ubuntu16.04-example
          name: mxnet
          resources: {}
  - PsRootPort: 9091
    mxReplicaType: WORKER
    replicas: 2
    template:
      metadata:
        creationTimestamp: null
      spec:
        containers:
        - args:
          - /incubator-mxnet/example/image-classification/train_mnist.py
          - --num-epochs
          - "15"
          - --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:1.4.1-gpu-py36-cu100-ubuntu16.04-example
          name: mxnet
          resources:
            limits:
              nvidia.com/gpu: 1
        restartPolicy: OnFailure
  terminationPolicy:
    chief:
      replicaIndex: 0
      replicaName: WORKER
status:
  phase: Running
  reason: ""
  replicaStatuses:
  - ReplicasStates:
      Running: 1
    mx_replica_type: SCHEDULER
    state: Running
  - ReplicasStates:
      Running: 2
    mx_replica_type: SERVER
    state: Running
  - ReplicasStates:
      Running: 2
```

```
    mx_replica_type: WORKER
    state: Running
  state: Running
```

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)”、“Failed (失败)”、“Done (完成)”之一。
状态 - 提供任务的总体状态且将为“Running (正在运行)”、“Succeeded (成功)”、“Failed (失败)”之一。

5. 清除并重新运行任务：

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

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：

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

MXNet 以及 Horovod 分布式 GPU 训练

本教程介绍如何在使用 Horovod 的多节点 GPU 集群上设置 MXNet 模型的分布式训练。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。本教程在 MNIST 模型上运行适用于 MXNet 的 Horovod 示例脚本 。

1. 设置应用程序名称并将其初始化。

```
$ APP_NAME=kubeflow-mxnet-hvd; ks init ${APP_NAME}; cd ${APP_NAME}
```

2. 在此应用程序的文件夹中的 kubeflow 安装 mpi-operator。

```
$ KUBEFLOW_VERSION=v0.5.1
$ ks registry add kubeflow github.com/kubeflow/kubeflow/tree/${KUBEFLOW_VERSION}/
kubeflow
$ ks pkg install kubeflow/common@${KUBEFLOW_VERSION}
$ ks pkg install kubeflow/mpi-job@${KUBEFLOW_VERSION}
$ ks generate mpi-operator mpi-operator
$ ks param set mpi-operator image mpioperator/mpi-operator:0.2.0
$ ks apply default -c mpi-operator
```

3. 创建 MPI 作业模板并定义节点数量（副本）以及每个节点具有的 GPU 数量 (gpusPerReplica)。您还可以使用您的映像并自定义命令。

```
IMAGE="763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-mxnet-training:1.6.0-
gpu-py36-cu101-ubuntu16.04-example"
GPUS_PER_WORKER=4
NUMBER_OF_WORKERS=3
JOB_NAME=mx-mnist-horovod-job
ks generate mpi-job-custom ${JOB_NAME}

ks param set ${JOB_NAME} replicas ${NUMBER_OF_WORKERS}
ks param set ${JOB_NAME} gpusPerReplica ${GPUS_PER_WORKER}
ks param set ${JOB_NAME} image ${IMAGE}
```

```
ks param set ${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,python,/horovod/examples/mxnet_mnist.py"
ks param set ${JOB_NAME} args -- --epochs=10,--lr=0.001
```

4. 检查创建的任务清单以验证一切内容是否显示正常。

```
$ ks show default -c ${JOB_NAME}
```

5. 将清单应用于默认环境。MPI 任务将创建启动 pod 并且日志将在此 pod 中聚合。

```
$ ks apply default -c ${JOB_NAME}
```

6. 检查状态。作业“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 -o wide
```

您应该可以看到类似于如下输出的内容：

```
NAME                                    READY     STATUS    RESTARTS   AGE       IP
            NODE                                NOMINATED NODE
mpi-operator-5fff9d76f5-wvf56           1/1       Running   0          23m
 192.168.10.117   ip-192-168-22-21.ec2.internal    <none>
mx-mnist-horovod-job-launcher-d7w6t     1/1       Running   0          21m
 192.168.13.210   ip-192-168-4-253.ec2.internal    <none>
mx-mnist-horovod-job-worker-0           1/1       Running   0          22m
 192.168.17.216   ip-192-168-4-253.ec2.internal    <none>
mx-mnist-horovod-job-worker-1           1/1       Running   0          22m
 192.168.20.228   ip-192-168-27-148.ec2.internal   <none>
mx-mnist-horovod-job-worker-2           1/1       Running   0          22m
 192.168.11.70    ip-192-168-22-21.ec2.internal    <none>
```

7. 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 mx-mnist-horovod-job-launcher-d7w6t
```

8. 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。
9. 清除并重新运行任务：

```
# make sure ${JOB_NAME} and ${APP_NAME} are still set
$ ks delete default -c ${JOB_NAME}
$ ks delete default
$ cd .. && rm -rf ${APP_NAME}
```

后续步骤

要了解有关将 MXNet 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅MXNet GPU 推理 (p. 60)。

TensorFlow 以及 Horovod 分布式 GPU 训练

本教程介绍如何在使用 Horovod 的多节点 GPU 集群上设置 TensorFlow 模型的分布式训练。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. 设置应用程序名称并将其初始化。

```
$ APP_NAME=kubeflow-tf-hvd; ks init ${APP_NAME}; cd ${APP_NAME}
```

2. 在此应用程序的文件夹中的 kubeflow 安装 mpi-operator。

```
$ KUBEFLOW_VERSION=v0.5.1
$ ks registry add kubeflow github.com/kubeflow/kubeflow/tree/${KUBEFLOW_VERSION}/
kubeflow
$ ks pkg install kubeflow/common@${KUBEFLOW_VERSION}
$ ks pkg install kubeflow/mpi-job@${KUBEFLOW_VERSION}
$ ks generate mpi-operator mpi-operator
$ ks param set mpi-operator image mpioperator/mpi-operator:0.2.0
$ ks apply default -c mpi-operator
```

3. 创建 MPI 任务模板并定义节点数量（副本）以及每个节点具有的 GPU 数量 (gpusPerReplica)，您还可以采用您的映像并自定义命令。

```
IMAGE="763104351884.dkr.ecr.us-east-1.amazonaws.com/aws-samples-tensorflow-
training:1.14.0-gpu-py36-cu100-ubuntu16.04-example"
GPUS_PER_WORKER=2
NUMBER_OF_WORKERS=3
JOB_NAME=tf-resnet50-horovod-job
ks generate mpi-job-custom ${JOB_NAME}

ks param set ${JOB_NAME} replicas ${NUMBER_OF_WORKERS}
ks param set ${JOB_NAME} gpusPerReplica ${GPUS_PER_WORKER}
ks param set ${JOB_NAME} image ${IMAGE}

ks param set ${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,python,/deep-learning-models/models/
resnet/tensorflow/train_imagenet_resnet_hvd.py"
ks param set ${JOB_NAME} args -- --num_epochs=10,--synthetic
```

4. 检查创建的任务清单以验证一切内容是否显示正常。

```
$ ks show default -c ${JOB_NAME}
```

5. 现在，将清单应用于默认环境。MPI 任务将创建启动 pod 并且日志将在此 pod 中聚合。

```
$ ks apply default -c ${JOB_NAME}
```

6. 检查状态。作业“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 -o wide
```

您应该可以看到类似于如下输出的内容：

```
NAME                                     READY     STATUS    RESTARTS   AGE       IP
            NODE                                NOMINATED NODE
mpi-operator-5fff9d76f5-wvf56            1/1       Running   0          23m
 192.168.10.117   ip-192-168-22-21.ec2.internal    <none>
tf-resnet50-horovod-job-launcher-d7w6t   1/1       Running   0          21m
 192.168.13.210   ip-192-168-4-253.ec2.internal    <none>
tf-resnet50-horovod-job-worker-0         1/1       Running   0          22m
 192.168.17.216   ip-192-168-4-253.ec2.internal    <none>
```

```
tf-resnet50-horovod-job-worker-1         1/1       Running   0          22m
 192.168.20.228   ip-192-168-27-148.ec2.internal   <none>
tf-resnet50-horovod-job-worker-2         1/1       Running   0          22m
 192.168.11.70    ip-192-168-22-21.ec2.internal    <none>
```

7. 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 tf-resnet50-horovod-job-launcher-d7w6t
```

8. 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。
9. 清除并重新运行任务：

```
# make sure ${JOB_NAME} and ${APP_NAME} are still set
$ ks delete default -c ${JOB_NAME}
$ ks delete default
$ cd .. && rm -rf ${APP_NAME}
```

后续步骤

要了解有关将 TensorFlow 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅TensorFlow GPU 推理 (p. 62)。

PyTorch 分布式 GPU 训练

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. 设置命名空间。

```
$ NAMESPACE=pytorch-multi-node-training; kubectl create namespace ${NAMESPACE}
```

2. 设置应用程序名称并将其初始化。

```
$ APP_NAME=eks-pytorch-mnist-app; ks init ${APP_NAME}; cd ${APP_NAME}
```

3. 将默认环境使用的命名空间更改为 `${NAMESPACE}`。

```
$ ks env set default --namespace ${NAMESPACE}
```

4. 为 kubeflow 安装 pytorch 运算符。这是使用参数服务器运行 PyTorch 分布式训练所需的。

```
$ export KUBEFLOW_VERSION=0.6.1
$ ks registry add kubeflow github.com/kubeflow/kubeflow/tree/v${KUBEFLOW_VERSION}/
kubeflow
$ ks pkg install kubeflow/pytorch-job@v${KUBEFLOW_VERSION}
```

5. 生成与 Kubernetes 兼容的 jsonnet 组件清单文件。

```
$ ks generate pytorch-operator pytorch-operator
```

6. 应用配置设置。

```
$ ks apply default -c pytorch-operator
```

7. 使用自定义资源定义 (CRD) 将使用户能够创建和管理 PyTorch 作业，就像内置的 K8s 资源一样。验证是否已安装 PyTorch 自定义资源。

```
$ kubectl get crd
```

8. 应用 nvidia 插件。

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

9. 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 `distributed.yaml` 的文件中。

```
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/pytorch-
training:1.3.1-gpu-py36-cu101-ubuntu16.04"
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
    Worker:
      replicas: 2
      restartPolicy: OnFailure
      template:
        spec:
          containers:
          - name: "pytorch"
            image: "763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-
training:1.3.1-gpu-py36-cu101-ubuntu16.04 "
            args:
              - "--backend"
              - "gloo"
              - "--epochs"
              - "5"
            resources:
              limits:
                nvidia.com/gpu: 1
```

10. 使用您刚创建的 pod 文件运行分布式训练任务。

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

11. 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。

后续步骤

要了解有关将 PyTorch 与 Deep Learning Containers 结合使用在 Amazon EKS 上进行基于 GPU 的推理，请参阅PyTorch GPU 推理 (p. 66)。

推理

本部分将指导您如何使用 MXNet、PyTorch、TensorFlow 和 TensorFlow 2 在 EKS 的 AWS Deep Learning Containers上运行推理。推理使用 CPU 或 GPU 集群，但仅限于单节点配置。

目录

CPU 推理

本部分介绍如何使用 MXNet、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Deep Learning Containers 映像 (p. 72)。

> **Note**
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

> **Note**
>
> 对于以下针对 MXNet 的基于 CPU 和 GPU 的推理示例，预训练 squeezenet 模型是公开的，因此，您无需修改提供的 yaml 文件。TensorFlow 示例将让您下载该模型，上传到您选择的 S3 存储桶，使用您的 AWS CLI 安全设置修改提供的 yaml 文件，然后使用修改过的 yaml 文件进行推理。

目录

MXNet CPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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：

```
$ 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 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。不要忘记将密钥应用于您的命名空间：

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

3. 创建文件 `mx_inference.yaml`。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-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 的状态并等待 pod 处于“RUNNING (正在运行)”状态：

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

6. 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
```

7. 要进一步描述 pod，您可以运行：

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

8. 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 &
```

9. 下载小猫的图像：

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

10. 在该模型上运行推理：

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

TensorFlow CPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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：

```
$ NAMESPACE=mx-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、S3 等。由于该服务需要访问 S3 和 ECR，您必须将您的 AWS 凭证存储为 Kubernetes 密钥。此项将设置为您之前安装的 tf-serving 程序包的参数。在本示例中，您将使用 S3 来存储和提取训练后的模型。

   检查您的 AWS 凭证。这些凭证必须具有 S3 写入访问权限。

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

3. 输出将类似于以下内容：

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

4. 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==
```

5. 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: BASE64OUTPUT==
  AWS_SECRET_ACCESS_KEY: BASE64OUTPUT==
```

6. 将密钥应用于您的命名空间：

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

7. 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为 `saved_model_half_plus_two` 的 S3 存储桶。

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

8. 同步 CPU 模型。

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

9. 创建文件 `tf_inference.yaml`。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-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 的状态并等待 pod 处于“RUNNING (正在运行)”状态：

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

12. 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
```

13. 要进一步描述 pod，您可以运行：

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

14. 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 &
```

15. 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]}
```

16. 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 CPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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：

```
$ 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 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。不要忘记将密钥应用于您的命名空间：

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

3. 创建文件 `pt_inference.yaml`。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-
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，您可以运行：

```
$ 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，（& 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 &
```

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. 68)。

GPU 推理

本部分介绍如何使用 MXNet、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Deep Learning Containers 映像 (p. 72)。

> Note
>
> MKL 用户：阅读AWS Deep Learning Containers Intel 数学内核库 (MKL) 建议 (p. 79)以获得最佳训练或推理性能。

目录

MXNet GPU 推理

在此方法中，创建一个 Kubernetes 服务和部署。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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。

```
$ 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。不要忘记将密钥应用于您的命名空间：

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

5. 创建文件 `mx_inference.yaml`。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。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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：

```
$ NAMESPACE=mx-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，您必须将您的 AWS 凭证存储为 Kubernetes 密钥。此项将设置为您之前安装的 tf-serving 程序包的参数。在本示例中，您将使用 S3 来存储和提取训练后的模型。

检查您的 AWS 凭证。这些凭证必须具有 S3 写入访问权限。

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

5. 输出将类似于以下内容：

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

6. 使用 base64 对这些凭证进行编码。首先编码访问密钥。

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

接下来编码秘密访问密钥。

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

您的输出应类似于以下内容：

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

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

```
apiVersion: v1
```

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

8. 将密钥应用于您的命名空间：

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

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

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

10. 同步 CPU 模型。

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

11. 创建文件 `tf_inference.yaml`。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。服务在其其他功能中公开了一个流程及其端口和部署，负责确保特定数量的 pod（在以下情况下，至少 1 个）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。不要忘记将密钥应用于您的命名空间。

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

5. 创建文件 `pt_inference.yaml`。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. 68)。

自定义入口点

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

更新 Pod 文件中的 `command` 参数。将 `args` 参数替换为您的自定义入口点脚本。

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-multi-model-server-densenet
```

```
spec:
  restartPolicy: OnFailure
  containers:
  - name: pytorch-multi-model-server-densenet
    image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.2.0-cpu-py36-
ubuntu16.04
    command:
      - "/bin/sh"
      - "-c"
    args:
      - /usr/local/bin/mxnet-model-server
      - --start
      - --mms-config /home/model-server/config.properties
      - --models densenet="https://dlc-samples.s3.amazonaws.com/pytorch/multi-model-server/
densenet/densenet.mar"
```

command 是 entrypoint 的 Kubernetes 字段名称。有关详细信息，请参阅此 Kubernetes 字段名称表。

如果 EKS 集群具有过期的 IAM 权限访问包含该映像的 ECR 存储库，或者您使用的 kubectl 来自与创建该集群的用户不同的用户，则您将收到以下错误消息。

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

要解决此问题，您需要刷新 IAM 令牌。请运行以下脚本。

```
set -ex

AWS_ACCOUNT=${AWS_ACCOUNT}
AWS_REGION=us-east-1
DOCKER_REGISTRY_SERVER=https://${AWS_ACCOUNT}.dkr.ecr.${AWS_REGION}.amazonaws.com
DOCKER_USER=AWS
DOCKER_PASSWORD=`aws ecr get-login --region ${AWS_REGION} --registry-ids ${AWS_ACCOUNT} |
 cut -d' ' -f6`
kubectl delete secret aws-registry || true
kubectl create secret docker-registry aws-registry \
--docker-server=$DOCKER_REGISTRY_SERVER \
--docker-username=$DOCKER_USER \
--docker-password=$DOCKER_PASSWORD
kubectl patch serviceaccount default -p '{"imagePullSecrets":[{"name":"aws-registry"}]}'
```

将 spec 下的以下内容附加到您的 Pod 文件中。

```
imagePullSecrets:
    - name: aws-registry
```

EKS 上的 AWS Deep Learning Containers 故障排除

故障排除

主题

设置错误

错误：错误。注册表 kubeflow 不存在

示例：

```
$ ks pkg install kubeflow/tf-serving
ERROR registry 'kubeflow' does not exist
```

解决方案：运行以下命令。

```
ks registry add kubeflow github.com/google/kubefl ow/tree/master/kubeflow
```

错误：上下文超出截止日期

示例：

```
$ 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
```

解决方案：验证是否超出您的账户的容量。在其他区域中重试该命令。

```
$ kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or
 port?
```

解决方案：尝试运行 `cp ~/.kube/eksctl/clusters/<name-of-cluster> ~/.kube/config`

```
$ ks apply default
ERROR handle object: patching object from cluster: merging object with existing state:
 Unauthorized
```

解决方案：这是一个并发问题，当具有不同授权/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。

```
$ 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
```

解决方案：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。您可能需要删除该文件夹以简化清理。

使用错误

```
ssh: Could not resolve hostname openmpi-worker-1.openmpi.kubeflow-dist-train-tf: Name or
 service not known
```

解决方案：在使用 EKS 集群时，只要看到此错误消息，则针对 Kubernetes 安装步骤再运行一遍 NVIDIA 设备插件。通过以下方式确保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

```
$ kubectl delete namespace ${NAMESPACE}
error: the server doesn't have a resource type "namspace"
```

解决方案：检查命名空间的拼写。可能拼写有错误。

```
$ ks delete default
ERROR the server has asked for the client to provide credentials
```

解决方案：确保 `~/.kube/config` 指向正确的集群，并且已使用 `aws configure` 或通过导出 AWS 环境变量正确配置了 AWS 凭证。

```
$ ks delete default
ERROR finding app root from starting path: : unable to find ksonnet project
$ kubectl logs -n ${NAMESPACE} -f ${COMPONENT}-master > results/benchmark_1.out
Error from server (NotFound): pods "openmpi-master" not found
```

解决方案：确保正确地更改目录到创建的 ksonnet 应用程序（执行 `ks init` 的文件夹），同时要注意，删除默认上下文将导致相应的资源被删除。

```
$ ks component rm openmpi
ERROR finding app root from starting path: : unable to find ksonnet project
```

解决方案：确保正确地更改目录到创建的 ksonnet 应用程序（执行 `ks init` 的文件夹）。

Deep Learning Containers 映像

AWS Deep 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。下表列出了任务定义中的 Amazon ECS 将使用的 Docker 映像 URL。

Deep Learning Containers Docker 映像在以下区域提供：

区域	代码	通用容器	Elastic Inference 容器
美国东部 (弗吉尼亚北部)	us-east-1	Available	Available
美国东部 (俄亥俄)	us-east-2	Available	Available
美国西部 (加利福尼亚北部)	us-west-1	Available	无
美国西部 (俄勒冈)	us-west-2	Available	Available
亚太地区 (东京)	ap-northeast-1	Available	Available
亚太地区 (首尔)	ap-northeast-2	Available	Available
亚太地区（香港）	ap-east-1	Available	无
亚太地区 (孟买)	ap-south-1	Available	无
亚太地区 (新加坡)	ap-southeast-1	Available	无
亚太地区 (悉尼)	ap-southeast-2	Available	无
中东（巴林）	me-south-1	Available	无
南美洲（圣保罗）	sa-east-1	Available	无
加拿大 (中部)	ca-central-1	Available	无
欧洲 (法兰克福)	eu-central-1	Available	无
欧洲（斯德哥尔摩）	eu-north-1	Available	无
欧洲 (爱尔兰)	eu-west-1	Available	Available
欧洲 (伦敦)	eu-west-2	Available	无
欧洲 (巴黎)	eu-west-3	Available	无

注意：eu-north-1 从 2.0 发行版本开始可用。因此，MXNet-1.4.0 在此区域中不可用。

ECR 是一项区域性服务并且映像表包含 us-east-1 映像的 URL。要从之前提到的区域之一拉取映像，请将该区域插入此示例后面的存储库 URL 中：

```
763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-training:1.15.2-cpu-py27-ubuntu18.04
```

Important

要拉取映像，必须先登录以访问 Deep Learning Containers 映像存储库。在以下命令中指定您的区域：

```
$(aws ecr get-login --no-include-email --region us-east-1 --registry-ids
 763104351884)
```

然后，运行以下命令，从 Amazon ECR 中拉取这些 Docker 映像：

```
docker pull <name of container image>
```

通用框架容器

要使用下表，请选择您所需的框架，以及您正在开始的作业类型和所需的 Python 版本。您的作业类型为 `training` 或 `inference`。您的 Python 版本为 `py27` 或 `py36`。如示例 URL 所示，将此信息插入 URL 的可替换部分。

Note

如果您使用 ap-east-1 区域，请将账户 ID“763104351884”替换为“871362719292”。如果您使用 me-south-1 区域，请将账户 ID“763104351884”替换为“217643126080”。

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
TensorFlow 2.1	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.1.0-cpu-*py27*-ubuntu18.04
TensorFlow 2.1	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.1.0-gpu-*py27*-cu101-ubuntu18.04
TensorFlow 2.0.1	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.0.1-cpu-*py27*-ubuntu18.04
TensorFlow 2.0.1	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.0.1-gpu-*py27*-cu100-ubuntu18.04

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
TensorFlow 1.15.2	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.15.2-cpu-*py27*-ubuntu18.04
TensorFlow 1.15.2	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.15.2-gpu-*py27*-cu100-ubuntu18.04
MXNet 1.6.0	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/mxnet-*training*:1.6.0-cpu-*py27*-ubuntu16.04
MXNet 1.6.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/mxnet-*training*:1.6.0-gpu-*py27*-cu101-ubuntu16.04
PyTorch 1.4.0	训练、推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.4.0-cpu-*py27*-ubuntu16.04
PyTorch 1.4.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.4.0-gpu-*py27*-cu101-ubuntu16.04

Elastic Inference 容器

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
带弹性推理的 TensorFlow 1.14.0	推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-inference-eia:1.14.0-cpu-*py27*-ubuntu16.04
带弹性推理的 MXNet 1.4.1	推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/mxnet-inference-eia:1.4.1-cpu-*py27*-ubuntu16.04

先前版本

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
TensorFlow 2.0.0	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.0.0-cpu-*py27*-ubuntu18.04
TensorFlow 2.0.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:2.0.0-gpu-*py27*-cu100-ubuntu18.04
TensorFlow 1.15.0	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.15.0-cpu-*py27*-ubuntu18.04
TensorFlow 1.15.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.15.0-gpu-*py27*-

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
					cu100-ubuntu18.04
TensorFlow 1.14.0	训练、推理	训练：是，推理：否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.14.0-cpu-*py27*-ubuntu16.04
TensorFlow 1.14.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/tensorflow-*training*:1.14.0-gpu-*py27*-cu100-ubuntu16.04
MXNet 1.4.1	训练、推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/mxnet-*training*:1.4.1-cpu-*py27*-ubuntu16.04
MXNet 1.4.1	训练、推理	否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/mxnet-*training*:1.4.1-gpu-*py27*-cu100-ubuntu16.04
PyTorch 1.3.1	训练、推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.3.1-cpu-*py27*-ubuntu16.04
PyTorch 1.3.1	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.3.1-gpu-*py27*-cu101-ubuntu16.04
PyTorch 1.2.0	训练、推理	否	CPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.2.0-cpu-*py27*-ubuntu16.04

Framework	作业类型	Horovod 选项	CPU/GPU	Python 版本选项	示例 URL
PyTorch 1.2.0	训练、推理	训练：是，推理：否	GPU	2.7 (py27)，3.6 (py36)	763104351884.dkr.ecr.*us-east-1*.amazonaws.com/pytorch-*training*:1.2.0-gpu-*py27*-cu100-ubuntu16.04

Deep Learning Containers 资源

以下主题介绍其他 AWS Deep Learning Containers 资源。

目录

构建 AWS Deep Learning Containers 自定义映像

如何构建自定义映像

我们可以轻松利用 Deep Learning Containers 自定义训练和推理，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 并构建它，以支持适用于多节点分布式训练的 Horovod 和 Python 3。它还将具有 AWS 示例 GitHub 存储库，该库包含许多深度学习模型示例。

```
#Take base container
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:1.15.2-gpu-py36-
cu100-ubuntu18.04

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

利用 MXNet 进行训练

在以下示例 Dockerfile 中，生成的 Docker 映像将构建针对 GPU 推理而优化的 MXNet v1.6.0，以支持 Horovod 和 Python 3。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 映像存储库。在下面的命令中指定您的区域和注册表 ID：

```
$(aws ecr get-login --no-include-email --region us-east-1 --registry-
ids YOUR_AWS_ACCOUNT_ID)
```

AWS Deep Learning Containers Intel 数学内核库 (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 AWS EC2 c5.18xlarge 实例上，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
```

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"
}
```

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
```

AWS Deep Learning Containers 中的安全性

AWS 的云安全性的优先级最高。作为 AWS 客户，您将从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 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 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。作为 AWS 合规性计划的一部分，第三方审计人员将定期测试和验证安全性的有效性。要了解适用于 Deep Learning Containers 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性 – 您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

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 Deep 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 AWS 服务来帮助您监控和保护 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 SageMaker 中的安全性。

主题

AWS Deep Learning Containers 中的数据保护

AWS Deep Learning Containers 符合 AWS 责任共担模式，此模式包含适用于数据保护的法规和准则。AWS 负责保护运行所有 AWS 服务的全球基础设施。AWS 保持对此基础设施上托管的数据的控制，包括用于处理客户内容和个人数据的安全配置控制。充当数据控制者或数据处理者的 AWS 客户和 APN 合作伙伴对他们在 AWS 云中放置的任何个人数据承担责任。

出于数据保护的目的，我们建议您保护 AWS 账户凭证并使用 AWS Identity and Access Management (IAM) 设置单个用户账户，以便仅向每个用户提供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 AWS 资源进行通信。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 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 Amazon S3 中的个人数据。

我们强烈建议您切勿将敏感的可识别信息（例如您客户的账号）放入自由格式字段（例如 Name (名称) 字段）。这包括使用控制台、API、AWS CLI 或 AWS 开发工具包处理 Deep Learning Containers 或其他 AWS

服务时。您输入到 Deep Learning Containers 或其他服务中的任何数据都可能被选取以包含在诊断日志中。当您向外部服务器提供 URL 时，请勿在 URL 中包含凭证信息来验证您对该服务器的请求。

有关数据保护的更多信息，请参阅 Amazon EC2 中的数据保护、Amazon SageMaker 中的数据保护、AWS 安全性博客上的 AWS 责任共担模型和 GDPR 博客文章。

AWS Deep Learning Containers 中的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（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 AWS 服务。

有关 Identity and Access Management 的更多信息，请参阅适用于 Amazon EC2 的 Identity and Access Management、适用于 Amazon ECS 的 Identity and Access Management、适用于 Amazon EKS 的 Identity and Access Management 和适用于 Amazon SageMaker 的 Identity and Access Management。

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 AWS 的方法。有关使用 AWS 管理控制台登录的更多信息，请参阅 IAM 用户指南 中的 IAM 控制台和登录页面。

您必须以 AWS 账户根用户、IAM 用户身份或通过代入 IAM 角色进行身份验证（登录到 AWS）。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。在这些案例中，您的管理员以前使用 IAM 角色设置了联合身份验证。在您使用来自其他公司的凭证访问 AWS 时，您间接地代入了角色。

要直接登录到 AWS 管理控制台，请使用您的密码和 根用户 电子邮件或 IAM 用户名。您可以使用 根用户 或 IAM 用户访问密钥以编程方式访问 AWS。AWS 提供了开发工具包和命令行工具，可使用您的凭证对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。使用签名版本 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 AWS General Reference 中的签名版本 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他安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅 IAM 用户指南 中的在 AWS 中使用多重身份验证 (MFA)。

AWS 账户根用户

当您首次创建 AWS 账户时，最初使用的是一个对账户中所有 AWS 服务和资源有完全访问权限的单点登录身份。此身份称为 AWS 账户 根用户，可使用您创建账户时所用的电子邮件地址和密码登录来获得此身份。强烈建议您不使用 根用户 执行日常任务，即使是管理任务。请遵守仅将 根用户 用于创建首个 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 用户和群组

IAM 用户是 AWS 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 用户可以拥有长期凭证，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 是指定一个 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，您有一个名为 IAMAdmins 的组并为该组授予管理 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 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 AWS 账户中具有特定权限的实体。它类似于 IAM 用户，但未与特定人员关联。您可以通过切换角色，在 AWS 管理控制台中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义 URL 以代入角色。有关使用角色方法的更多信息，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 用户可代入 IAM 角色，暂时获得针对特定任务的不同权限。
- 联合身份用户访问 – 您也可以不创建 IAM 用户，而是使用来自 AWS 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身份用户。在通过身份提供商请求访问权限时，AWS 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 角色允许其他账户中的某个人（可信任委托人）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 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南 中的 IAM 角色与基于资源的策略有何不同。
- AWS 服务访问 –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。在设置一些 AWS 服务环境时，您必须为服务定义要代入的角色。这个服务角色必须包含该服务访问所需的 AWS 资源会用到的所有必要权限。服务角色因服务而异，但只要您满足服务记录在案的要求，许多服务都允许您选择权限。服务角色只在您的账户内提供访问权限，不能用于为访问其他账户中的服务授权。您可以在 IAM 中创建、修改和删除服务角色。例如，您可以创建一个角色，此角色允许 Amazon Redshift 代表您访问 Amazon S3 存储桶，然后将该存储桶中的数据加载到 Amazon Redshift 集群中。有关更多信息，请参阅 IAM 用户指南 中的创建角色以向 AWS 服务委派权限。
- 在 Amazon EC2 上运行的应用程序–对于在 EC2 实例上运行、并发出 AWS CLI 或 AWS API 请求的应用程序，您可以使用 IAM 角色管理它们的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 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 角色向在 Amazon EC2 实例上运行的应用程序授予权限。

要了解是否使用 IAM 角色，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 AWS 资源，以便控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在某个实体（根用户、IAM 用户或 IAM 角色）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 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 策略概述。

IAM 管理员可以使用策略来指定哪些用户有权访问 AWS 资源，以及他们可以对这些资源执行哪些操作。每个 IAM 实体（用户或角色）在一开始都没有权限。换言之，默认情况下，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 策略定义操作的权限，无论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 AWS 管理控制台、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、角色或组）的 JSON 权限策略文档。这些策略控制身份可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南 中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是可以附加到 AWS 账户中的多个用户、组和角色的独立策略。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管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 (如 Amazon S3 存储桶) 的 JSON 策略文档。服务管理员可以使用这些策略来定义指定的委托人（账户成员、用户或角色）可以对该资源以及在什么条件执行哪些操作。基于资源的策略是内联策略。没有基于托管资源的策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 是一种策略类型，用于控制哪些委托人（账户成员、用户或角色）有权访问资源。ACL 类似于基于资源的策略，但它们不使用 JSON 策略文档格式。Amazon S3、AWS 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) 概述。

其他策略类型

AWS 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 – 权限边界是一项高级功能，借助该功能，您可以设置基于身份的策略可以授予 IAM 实体的最大权限（IAM 用户或角色）。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 AWS Organizations 中的最大权限。AWS Organizations 是一项服务，用于分组和集中管理您的企业拥有的多个 AWS 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（包括每个 AWS 账户根用户）的权限。有关 组织 和 SCP 的更多信息，请参阅 AWS Organizations 用户指南 中的 SCP 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 AWS 如何确定在涉及多个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 替换为 Amazon EMR

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

有关将 IAM 与 Amazon EMR 结合使用的更多信息，请参阅 Amazon EMR 的 AWS Identity and Access Management。

AWS Deep Learning Containers 中的日志记录和监控

AWS Deep 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 EKS 和监控 Amazon SageMaker。

使用情况跟踪

以下 Deep Learning Containers 包含允许 AWS 收集用于容器的实例类型的代码。不会收集有关所用命令的信息或其他任何关于容器的信息。

- TensorFlow 1.15
- TensorFlow 2.0
- 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

AWS Deep Learning Containers 的合规性验证

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

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

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

您在使用 Deep Learning Containers 时的合规性责任由您数据的敏感性、贵公司的合规性目标以及适用的法律法规决定。AWS 提供以下资源来帮助满足合规性：

- 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 AWS 上部署基于安全性和合规性的基准环境的步骤。
- AWS 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。
- AWS Config 开发人员指南中的使用规则评估资源 – 此 AWS 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。
- AWS Security Hub – 此 AWS 服务提供了 AWS 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。

AWS Deep Learning Containers 中的弹性

AWS 全球基础设施围绕 AWS 区域和可用区构建。AWS 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 AWS 区域和可用区的更多信息，请参阅 AWS 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 中的弹性、Amazon EKS 中的弹性和 Amazon SageMaker 中的弹性。

AWS Deep Learning Containers 中的基础设施安全性

AWS Deep Learning Containers 的基础架构安全性受 Amazon EC2、Amazon ECS、Amazon EKS 或 SageMaker 支持。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 SageMaker 中的基础设施安全性。

针对 Deep Learning Containers 开发人员指南的文档历史记录

下表介绍此 Deep Learning Containers 发布版本的文档。

- API 版本：最新
- 最近文档更新时间：Month DD, YYYY

update-history-change	update-history-description	update-history-date
Deep Learning Containers 开发人员指南启动 (p. 1)	Deep Learning Containers 设置和教程已添加到开发人员指南中。	February 17, 2020

AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 AWS 词汇表。