亚马逊云科技

开发人员指南

Amazon Deep Learning AMIs

Amazon Deep Learning AMIs: 开发人员指南

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

Table of Contents

什么是 Amazon Deep Learning AMIs？

Amazon Deep Learning AMIs (DLAMI) 提供自定义的机器映像，可用于云端的深度学习。 DLAMIs 它们大多 Amazon Web Services 区域 适用于各种亚马逊弹性计算云 (Amazon EC2) 实例类型，从仅限 CPU 的小型实例到最新的高性能多 GPU 实例。它们预 DLAMIs 先配置了 NVIDIA CUDA 和 NVIDIA cuDNN 以及最受欢迎的深度学习框架的最新版本。

关于本指南

中的内容可以帮助您启动和使用 DLAMIs。该指南涵盖了几个可同时用于训练和推理的常见深度学习使用案例。还介绍了如何针对您的用途选择合适的 AMI 以及您可能喜欢的实例类型。

此外，还 DLAMIs 包括其支持的框架提供的几个教程。本指南可以向您展示如何激活每个框架，并找到相应的入门教程。它还有关于分布式训练、调试、使用 Amazon Inferentia 和 Amazon Trainium 以及其他关键概念的教程。有关如何设置 Jupyter Notebook 服务器以在浏览器中运行教程的说明，请参阅在 DLAMI 实例上设置 Jupyter Notebook 服务器。

先决条件

要成功运行 DLAMIs，我们建议您熟悉命令行工具和基本的 Python。

DLAMI 使用案例示例

以下是 Amazon Deep Learning AMIs (DLAMI) 的一些常见用例的示例。

了解深度学习：DLAMI 是学习或教授机器学习和深度学习框架的理想选择。这 DLAMIs 消除了对每个框架的安装进行故障排除并让它们在同一台计算机上运行所带来的麻烦。 DLAMIs 其中包括 Jupyter 笔记本，便于运行框架为机器学习和深度学习新手提供的教程。

应用程序开发：如果您是应用程序开发人员，有兴趣使用深度学习来使应用程序利用 AI 领域的最新进展，则 DLAMI 是理想的测试平台。每个框架都附带了关于如何开始使用深度学习的教程，其中许多教程都有模型动物园，可以让您轻松试用深度学习，您不需要自己创建神经网络或进行模型训练。一些示例向您展示如何在几分钟内构建映像检测应用程序，或如何为您自己的聊天自动程序构建语音识别应用程序。

机器学习和数据分析：如果您是数据科学家，或者您有兴趣通过深度学习处理数据，则您会发现许多框架都支持 R 和 Spark。您将找到关于进行简单回归的教程，以及为个性化和预测系统构建可扩展的数据处理系统的教程。

研究 — 如果你是一名想要尝试新框架、测试新模型或训练新模型的研究人员，那么 DLAMI Amazon 和扩展能力可以减轻繁琐的安装和管理多个训练节点所带来的痛苦。

> **Note**
>
> 虽然您最初的选择可能是将您的实例类型升级到具有更多实例的大型实例 GPUs（最多 8 个），但您也可以通过创建 DLAMI 实例集群进行横向扩展。有关集群构建的更多信息，请参阅 有关 DLAMI 的相关信息。

DLAMI 的功能

Amazon Deep Learning AMIs (DLAMI) 的功能包括预装的深度学习框架、GPU 软件、模型服务器和模型可视化工具。

预装的框架

目前有两种主要的 DLAMI 版本，均包含与操作系统（OS）和软件版本相关的其它变体：

- 带 Conda 的深度学习 AMI：在单独的 Python 环境中使用 conda 程序包单独安装的框架。
- 深度学习基础 AMI：不安装任何框架；只有 NVIDIA CUDA 和其它依赖项。

带 Conda 的深度学习 AMI 使用 conda 环境来隔离每个框架，以便您可以随意切换框架，而不用担心其依赖项发生冲突。带 Conda 的 Deep Learning AMI 支持以下框架：

- PyTorch
- TensorFlow 2

> **Note**
>
> DLAMI 不再支持以下深度学习框架：Apache MXNet、微软认知工具包 (CNTK)、Caffe、Caffe、Caffe2、Theano、Chainer 和 Keras。

预装的 GPU 软件

即使你使用仅限 CPU 的实例，它们 DLAMIs 也会有 NVIDIA CUDA 和 NVIDIA cu DNN。无论实例类型是什么，安装的软件都是相同的。请记住，GPU 特定的工具只适用于至少具有一个 GPU 的实例。有关实例类型的更多信息，请参阅选择 DLAMI 实例类型。

有关 CUDA 的更多信息，请参阅 CUDA 安装和框架绑定。

模型处理和可视化

带有 Conda 的 Deep Learning AMI 预装了用于 TensorFlow模型可视化和模型 TensorBoard可视化的模型服务器。有关更多信息，请参阅 TensorFlow 服务。

的发行说明 DLAMIs

在这里，您可以找到当前支持的所有选项 Amazon Deep Learning AMIs (DLAMI) 的详细发行说明。

有关我们不再支持的 DLAMI 框架的发布说明，请参阅 DLAMI Framework Support Policy 页面的 Unsupported Framework Release Notes Archive 部分。

> **Note**
>
> Amazon Deep Learning AMIs 他们有每晚发布安全补丁的节奏。我们不会在官方发布说明中包含这些增量安全补丁。

基地 DLAMIs

GPU

- X86
  - Amazon 深度学习基础 AMI（亚马逊 Linux 2023）
  - Amazon 深度学习基础 AMI (Ubuntu 22.04)
  - Amazon 深度学习基础 AMI (Ubuntu 20.04)
  - Amazon 深度学习基础 AMI（亚马逊 Linux 2）
- ARM64
  - Amazon 深度学习基础 ARM64 AMI (Ubuntu 22.04)
  - Amazon 深度学习基础 ARM64 AMI（亚马逊 Linux 2）
  - Amazon 深度学习基础 ARM64 AMI（亚马逊 Linux 2023）

Qualcomm

- X86
  - Amazon 深度学习基础高通 AMI（亚马逊 Linux 2）

Amazon Neuron

- 请参阅 Neuron DLAMI 指南

单一框架 DLAMIs

PyTorch-特定的 AMIs

GPU

- X86
  - Amazon 深度学习 AMI GPU PyTorch 2.6（亚马逊 Linux 2023）
  - Amazon 深度学习 AMI GPU PyTorch 2.6 (Ubuntu 22.04)
  - Amazon 深度学习 AMI GPU PyTorch 2.5（亚马逊 Linux 2023）
  - Amazon 深度学习 AMI GPU PyTorch 2.5 (Ubuntu 22.04)
  - Amazon 深度学习 AMI GPU PyTorch 2.4 (Ubuntu 22.04)
  - Amazon 深度学习 AMI GPU PyTorch 2.3 (Ubuntu 20.04)
  - Amazon 深度学习 AMI GPU PyTorch 2.3 (亚马逊 Linux 2)
- ARM64
  - Amazon 深度学习 ARM64 AMI GPU PyTorch 2.6（亚马逊 Linux 2023）
  - Amazon 深度学习 ARM64 AMI GPU PyTorch 2.6 (Ubuntu 22.04)
  - Amazon 深度学习 ARM64 AMI GPU PyTorch 2.5 (Ubuntu 22.04)
  - Amazon 深度学习 ARM64 AMI GPU PyTorch 2.4 (Ubuntu 22.04)
  - Amazon 深度学习 ARM64 AMI GPU PyTorch 2.3 (Ubuntu 22.04)

Amazon Neuron

- 请参阅 Neuron DLAMI 指南

TensorFlow-特定的 AMIs

GPU

- X86
  - Amazon 深度学习 AMI GPU TensorFlow 2.18（亚马逊 Linux 2023）
  - Amazon 深度学习 AMI GPU TensorFlow 2.18 (Ubuntu 22.04)
  - Amazon 深度学习 AMI GPU TensorFlow 2.17 (Ubuntu 22.04)

Amazon Neuron

- 请参阅 Neuron DLAMI 指南

多框架 DLAMIs

> **Tip**
>
> 如果您只使用一个机器学习框架，我们建议使用单框架 DLAMI。

GPU

- X86
  - Amazon 深度学习 AMI (Amazon Linux 2)

Amazon Neuron

- 请参阅 Neuron DLAMI 指南

DLAMI 入门

本指南包括了关于选择适合您的 DLAMI、选择适合您的使用案例和预算的实例类型的技巧，以及介绍您可能感兴趣的自定义设置的 有关 DLAMI 的相关信息。

如果您不熟悉使用 Amazon 或使用 Amazon EC2，请从带 Conda 的深度学习 AMI。如果您熟悉 Amazon EC2 和 Amazon EMR、Amazon EFS 或 Amazon S3 等其他 Amazon 服务，并且有兴趣将这些服务集成有关 DLAMI 的相关信息到需要分布式训练或推理的项目中，那么请查看是否适合您的用例。

我们建议您查看选择 DLAMI，以了解最适合您的应用程序的实例类型。

后续步骤

选择 DLAMI

选择 DLAMI

如 GPU DLAMI发行说明中所述，我们提供了一系列D LAMI选项。为了帮助您为自己的使用案例选择正确的 DLAMI，我们按开发映像的硬件类型或功能对映像进行分组。我们的顶层分组是：

- DLAMI 类型：基本、单框架、多框架 (Conda DLAMI)
- 计算架构：基于 x86、基于 arm64 的 Graviton Amazon
- 处理器类型：GPU、CP U、Inferentia、T raini um
- SDK：CUDA、Amazon N euron
- 操作系统：亚马逊 Linux、Ubuntu

本指南中的其他主题有助于进一步给您提供信息和进一步让您了解详情。

主题

- CUDA 安装和框架绑定
- 深度学习基础 AMI
- 带 Conda 的深度学习 AMI
- DLAMI 架构选项
- DLAMI 操作系统选项

后续步骤

带 Conda 的深度学习 AMI

CUDA 安装和框架绑定

虽然深度学习都非常先进，但每个框架都提供“稳定”版本。这些稳定版本可能不适用于最新的 CUDA 或 cuDNN 实施和功能。您的用例和所需功能可以帮助您选择框架。如果您不确定，就请使用最新的带 Conda 的深度学习 AMI。它为所有使用 CUDA 的框架提供了官方 `pip` 二进制文件，同时使用每个框架均支持的最新版本。如果您需要最新版本，并要自定义深度学习环境，就请使用深度学习基础 AMI。

如需进一步指导，请在 稳定版本与候选版本 上查看我们的指南。

选择带 CUDA 的 DLAMI

深度学习基础 AMI 具有所有可用的 CUDA 版本系列

带 Conda 的深度学习 AMI 具有所有可用的 CUDA 版本系列

> **Note**
>
> 我们不再将 CNTK、Caffe MXNet、Caffe2、Theano、Chainer 或 Keras Conda 环境包含在中。 Amazon Deep Learning AMIs

有关具体框架版本号，请参阅 的发行说明 DLAMIs。

选择这种 DLAMI 类型，或者使用 “下一步” 选项详细了解 DLAMIs 不同的类型。

选择一个 CUDA 版本并在附录中查看包含该版本的 DLAMIs 完整列表，或者使用 Ne xt Up 选项了解有关不同版本 DLAMIs 的更多信息。

后续步骤

深度学习基础 AMI

相关主题

- 有关在 CUDA 版本之间切换的说明，请参阅使用深度学习基础 AMI教程。

深度学习基础 AMI

深度学习基础 AMI 就像一张用于深度学习的空白画布。它包含您需要的一切，直到安装特定的框架，并具有您选择的 CUDA 版本。

为什么要选择基础 DLAMI

该 AMI 团队对于那些想要分享深度学习项目和建立最新版本的项目贡献者非常有用。适用于那些希望推出自己环境且有信心安装和使用最新 NVIDIA 软件的人员，从而他们可以专注于选择要安装的框架和版本。

选择这种 DLAMI 类型，或者使用 “下一步” 选项详细了解 DLAMIs 不同的类型。

后续步骤

使用 Conda 的 DLAMI

相关主题

- 使用深度学习基础 AMI

带 Conda 的深度学习 AMI

Conda DLAMI conda 使用虚拟环境，它们存在于多框架或单一框架中。 DLAMIs这些环境配置为单独安装不同的框架，并简化框架之间的切换。这对了解和体验 DLAMI 必须提供的所有框架很有好处。大多数用户都会发现新的带 Conda 的深度学习 AMI 非常适合他们。

它们将经常使用框架中的最新版本进行更新，并拥有最新的 GPU 驱动程序和软件。在大多数文档 Amazon Deep Learning AMIs 中，它们通常被称为。它们 DLAMIs 支持 Ubuntu 20.04、Ubuntu 22.04、亚马逊 Linux 2、亚马逊 Linux 2023 操作系统。操作系统支持取决于上游操作系统的支持。

稳定版本与候选版本

Conda AMIs 使用每个框架中最新正式版本的优化二进制文件。不会有候选版本和实验性功能。优化取决于框架对 Intel 的 MKL DNN 等加速技术的支持，这些技术可加快在 C5 和 C4 CPU 实例类型上的训练和推理速度。编译二进制文件还支持高级英特尔指令集，包括但不限于 AVX、AVX-2、 SSE4 .1 和 SSE4 .2。这些指令将加快 Intel CPU 架构上向量和浮点运算的速度。此外，对于 GPU 实例类型，使用最新官方版本支持的任何版本来更新 CUDA 和 cuDNN。

首次激活框架后，带有 Conda 的深度学习 AMI 会自动为您的 Amazon EC2 实例安装最优化的框架版本。有关更多信息，请参阅使用带 Conda 的深度学习 AMI。

如果要使用自定义或优化的版本选项从源安装，深度学习基础 AMI 可能是您更好的选择。

Python 2 弃用

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 版本是最后一个支持 Python 2 的版本。包含 Python 2 Conda 环境的 DLAMI 先前版本（v26、v25 等）继续可用。但是，只有在开源社区针对先前发布的 DLAMI 版本发布了安全修补程序时，我们才会在这些版本上提供关于 Python 2 Conda 环境的更新。带有最新版本 PyTorch 和框架的 DLAMI 版本不包含 Python 2 Conda 环境。 TensorFlow

CUDA 支持

具体 CUDA 版本号可以在 GPU DLAMI 发布说明中找到。

后续步骤

DLAMI 架构选项

相关主题

- 有关使用带 Conda 的深度学习 AMI 的教程，请参阅 使用带 Conda 的深度学习 AMI 教程。

DLAMI 架构选项

Amazon Deep Learning AMIs附带基于 x86 或基于 Arm64 的 Amazon Graviton2 架构。

有关 ARM64 GPU DLAMI 入门的信息，请参阅。 ARM64 DLAMI有关可用实例类型的更多详细信息，请参阅 选择 DLAMI 实例类型。

后续步骤

DLAMI 操作系统选项

DLAMI 操作系统选项

DLAMIs 在以下操作系统中提供。

- Amazon Linux 2
- Amazon Linux 2023
- Ubuntu 20.04
- Ubuntu 22.04

旧版本的操作系统在已弃用 DLAMIs版本上可用。有关 DLAMI 弃用的更多信息，请参阅 DLAMI 的弃用

在选择 DLAMI 之前，请评估您需要的实例类型并确定您的 Amazon 区域。

后续步骤

选择 DLAMI 实例类型

选择 DLAMI 实例类型

在更一般情况下，在为 DLAMI 选择实例类型时，请考虑以下几点。

- 如果您不熟悉深度学习，那么具有单个 GPU 的实例可能适合您的需求。
- 如果您注重预算，则可以使用仅含 CPU 的实例。
- 如果您希望优化深度学习模型推断的高性能和成本效益，则可以使用带有 Amazon Inferentia 芯片的实例。
- 如果您正在寻找具有基于 Arm64 的 CPU 架构的高性能 GPU 实例，则可以使用 G5g 实例类型。
- 对于高容量推理服务，具有大量内存的单个 CPU 实例或此类实例的集群可能是更好的解决方案。
- 如果您使用的是具有大量数据或较大批处理大小的大型模型，那么您需要具有更多内存的大型实例。您也可以将模型分发到一个集群 GPUs。您可能会发现，如果减小批处理大小，则使用内存较少的实例将是更好的选择。这可能会影响您的准确性和训练速度。
- 如果您有兴趣使用 NVIDIA 集体通信库 (NCCL) 来运行机器学习应用程序，且需要大规模的节点间通信，那么您可能需要使用 Elastic Fabric Adapter (EFA)。

有关实例的更多详细信息，请参阅EC2 实例类型EC2 。

以下主题提供有关实例类型注意事项的信息。

> **Important**
>
> 深度学习 AMIs 包括由 NVIDIA 公司开发、拥有或提供的驱动程序、软件或工具包。您同意仅在包含 NVIDIA 硬件的 Amazon EC2 实例上使用这些 NVIDIA 驱动程序、软件或工具包。

主题

- DLAMI 的定价

DLAMI 的定价

DLAMI 中包含的深度学习框架是免费的，每个都有自己的开源许可证。尽管 DLAMI 中包含的软件是免费的，但您仍然需要为底层的 Amazon EC2 实例硬件付费。

有些 Amazon EC2 实例类型被标记为免费。可以在其中一个免费实例上运行 DLAMI。这意味着，当您只使用该实例的容量时，使用 DLAMI 是完全免费的。如果您需要一个功能更强大、具有更多 CPU 内核、更多磁盘空间、更多 RAM 或一个或多个 RAM 的实例 GPUs，那么您需要一个不属于免费套餐实例类别的实例。

有关实例选择和定价的更多信息，请参阅 Amazon EC2 定价。

DLAMI 区域可用性

每个区域支持不同的实例类型范围，并且通常不同区域的实例类型的成本略有不同。 DLAMIs 并非在每个地区都可用，但可以复制 DLAMIs 到您选择的区域。有关更多信息，请参阅复制 AMI。请注意区域选择列表，并确保您选择一个靠近您或您客户的区域。如果您打算使用不止一个 DLAMI 并且可能创建一个集群，请确保为集群中的所有节点使用相同的区域。

有关区域的更多信息，请访问EC2 区域 。

后续步骤

推荐的 GPU 实例

推荐的 GPU 实例

我们推荐的 GPU 实例适用于大多数深度学习目的。在 GPU 实例上训练新模型比在 CPU 实例上更快。当您拥有多 GPU 实例或在多个实例上使用分布式训练时，您可以进行亚线性扩展。 GPUs

以下实例类型支持 DLAMI。有关 GPU 实例类型选项及其用途的信息，请参阅EC2 实例类型EC2 并选择加速计算。

> **Note**
>
> 应将模型大小作为选择实例的一个因素。如果模型超出了实例的可用 RAM，请为应用程序选择其它具有足够内存的实例类型。

DLAMI 实例提供了用于监控和优化 GPU 进程的工具。有关监控 GPU 进程的更多信息，请参阅 GPU 监控和优化。

有关使用 G5g 实例的特定教程，请参阅 ARM64 DLAMI。

后续步骤

推荐的 CPU 实例

推荐的 CPU 实例

无论您是在关注预算，了解深度学习，还是只是想运行一项预测服务，您在 CPU 类别中都有许多经济实惠的选项。某些框架利用了 Intel 的 MKL DNN，从而加快了在 C5（并非在所有区域中都可用）CPU 实例类型上的训练和推理速度。有关 CPU 实例类型的信息，请参阅EC2 实例类型EC2 并选择计算优化。

> **Note**
>
> 应将模型大小作为选择实例的一个因素。如果模型超出了实例的可用 RAM，请为应用程序选择其它具有足够内存的实例类型。

后续步骤

推荐的 Inferentia 实例

推荐的 Inferentia 实例

Amazon Inferentia 实例旨在为深度学习模型推理工作负载提供高性能和成本效益。具体而言，Inf2 实例类型使用 Amazon Inferentia 芯片和 Ne Amazon uron SDK，后者与流行的机器学习框架（例如和）集成。 TensorFlow PyTorch

客户使用 Inf2 实例之后，能够以最低的云端成本来运行大规模的机器学习推理应用程序，例如搜索、推荐引擎、计算机视觉、语音识别、自然语言处理、个性化和欺诈检测。

> **Note**
>
> 应将模型大小作为选择实例的一个因素。如果模型超出了实例的可用 RAM，请为应用程序选择其它具有足够内存的实例类型。

- Amazon EC2 Inf2 实例最多有 16 个 Amazon 推理芯片和 100 Gbps 的网络吞吐量。

有关开始使用 Amazon Inferentia 的更多信息 DLAMIs，请参阅。带有 DLAMI 的 Amazon 推理芯片

后续步骤

推荐的 Trainium 实例

推荐的 Trainium 实例

Amazon Trainium 实例旨在为深度学习模型推理工作负载提供高性能和成本效益。具体而言，Trn1 实例类型使用 T Amazon rainium 芯片和 Ne Amazon uron SDK，后者与流行的机器学习框架（例如和）集成。 TensorFlow PyTorch

客户使用 Trn1 实例之后，能够以最低的云端成本来运行大规模的机器学习推理应用程序，例如搜索、推荐引擎、计算机视觉、语音识别、自然语言处理、个性化和欺诈检测。

> **Note**
>
> 应将模型大小作为选择实例的一个因素。如果模型超出了实例的可用 RAM，请为应用程序选择其它具有足够内存的实例类型。

- Amazon EC2 Trn1 实例最多有 16 个 T Amazon rainium 芯片和 100 Gbps 的网络吞吐量。

设置 DLAMI 实例

选择 DLAMI 并选择要使用的亚马逊弹性计算云 (A EC2 mazon) 实例类型后，就可以设置新的 DLAMI 实例了。

如果您尚未选择 DLAMI EC2 和实例类型，请参阅。DLAMI 入门

主题

- 查找 DLAMI 的 ID
- 启动 DLAMI 实例
- 连接到 DLAMI 实例
- 在 DLAMI 实例上设置 Jupyter Notebook 服务器
- 清理 DLAMI 实例

查找 DLAMI 的 ID

每个 DLAMI 都有唯一标识符（ID）。当您使用 EC2 亚马逊控制台启动 DLAMI 实例时，您可以选择使用 DLAMI ID 来搜索要使用的 DLAMI。当您使用 Amazon Command Line Interface Amazon CLI() 启动 DLAMI 实例时，需要此 ID。

你可以使用 Amazon CLI 亚马逊或 Parameter Store 的命令找到你选择的 DL EC2 AMI 的 ID，该功能为。 Amazon Systems Manager有关安装和配置的说明 Amazon CLI，请参阅《Amazon Command Line Interface 用户指南》 Amazon CLI中的 “入门”。

Using Parameter Store

使用 ssm get-parameter 查找 DLAMI ID

在以下ssm get-parameter命令中，对于该`--name`选项，参数名称的格式为*`/aws/service/deeplearning/ami/$architecture/$ami_type/latest/ami-id`*。在这种名称格式中，*`architecture`*可以是**`x86_64`**或**`arm64`**。使用 DLAMI 名称并删除关键词 “深入”、“学习” 和 “ami” 来指定。*`ami_type`*可以在 的发行说明 DLAMIs中找到 AMI 名称。

> **Important**
>
> 要使用此命令，您使用的 Amazon Identity and Access Management (IAM) 委托人必须具有`ssm:GetParameter`权限。有关 IAM 主体的更多信息，请参阅《IAM 用户指南》中 IAM 角色的其他资源部分。

- ```
  aws ssm get-parameter --name /aws/service/deeplearning/ami/x86_64/base-oss-nvidia-driver-ubuntu-22.04/latest/ami-id  \
  --region us-east-1 --query "Parameter.Value" --output text
  ```

  该输出值应该类似于以下内容：

  ```
  ami-09ee1a996ac214ce7
  ```

  > **Tip**
  >
  > 对于目前支持的一些 DLAMI 框架，可以在 的发行说明 DLAMIs中找到更具体的示例 ssm get-parameter 命令。选择指向所选 DLAMI 的发布说明的链接，然后在发布说明中查找其 ID 查询。

Using Amazon EC2 CLI

使用 ec2 describe-images 查找 DLAMI ID

在以下 ec2 describe-images 命令中，对于筛选条件 Name=name 的值，输入 DLAMI 名称。可以为给定的框架指定发布版本，也可以通过将版本号替换为问号（？）来获取最新版本。

- ```
  aws ec2 describe-images --region us-east-1 --owners amazon \
  --filters 'Name=name,Values=Deep Learning Base OSS Nvidia Driver GPU AMI (Ubuntu 22.04) ????????' 'Name=state,Values=available' \
  --query 'reverse(sort_by(Images, &amp;CreationDate))[:1].ImageId' --output text
  ```

  该输出值应该类似于以下内容：

  ```
  ami-09ee1a996ac214ce7
  ```

> **Tip**
>
> 有关特定于您选择的 DLAMI 的示例 ec2 describe-images 命令，请参阅 的发行说明 DLAMIs。选择指向所选 DLAMI 的发布说明的链接，然后在发布说明中查找其 ID 查询。

后续步骤

启动 DLAMI 实例

启动 DLAMI 实例

找到要用于启动 DLAMI 实例的 DLAMI 的 ID 后，就可以启动实例了。要启动它，你可以使用 Amazon EC2 控制台或 Amazon Command Line Interface (Amazon CLI)。

> **Note**
>
> 对于此演练，我们可能会针对深度学习基本 OSS Nvidia 驱动程序 GPU AMI（Ubuntu 22.04）进行引用。即使选择其他 DLAMI，您也应能按照本指南进行操作。

EC2 console

> **Note**
>
> 要加快高性能计算（HPC）和机器学习应用程序的速度，可以使用 Elastic Fabric Adapter（EFA）启动 DLAMI 实例。有关具体说明，请参阅使用 EFA 启动 Amazon Deep Learning AMIs 实例。

1. 打开控制EC2 台。
2. 在最上面的导航栏 Amazon Web Services 区域 中记下你当前的状态。如果这不是您所需的区域，请在继续操作之前更改此选项。有关更多信息，请参阅中的EC2 区域。
3. 选择启动实例。
4. 为您的实例输入名称，然后选择适合您的 DLAMI。

a. 在 “我的” 中查找现有 DLAMI 或选择 “快速入 AMIs门”。

b. 按 DLAMI ID 进行搜索。浏览这些选项，然后选中您的选择。

5. 选择一个实例类型。可以在 的发行说明 DLAMIs中找到适用于 DLAMI 的建议实例系列。有关 DLAMI 实例类型的一般建议，请参阅选择 DLAMI 实例类型。

6. 选择启动实例。

Amazon CLI

- 要使用 Amazon CLI，您必须拥有要使用的 DLAMI 的 ID、 Amazon Web Services 区域 EC2 和实例类型以及您的安全令牌信息。然后，您可以使用ec2 run-instances Amazon CLI 命令启动实例。

  有关安装和配置的说明 Amazon CLI，请参阅《Amazon Command Line Interface 用户指南》Amazon CLI中的 “入门”。有关更多信息，包括命令示例，请参阅启动、列出和关闭 Amazon EC2 实例 Amazon CLI。

使用 Amazon EC2 控制台或启动实例后 Amazon CLI，等待实例准备就绪。这通常仅需要几分钟时间。您可以在 Amazon EC2 控制台中验证实例的状态。有关更多信息，请参阅亚马逊 EC2 用户指南中的亚马逊 EC2 实例状态检查。

后续步骤

连接到 DLAMI 实例

连接到 DLAMI 实例

在您启动 DLAMI 实例并且该实例正在运行之后，可以使用 SSH 从客户端（Windows、macOS 或 Linux）连接到该实例。有关说明，请参阅亚马逊 EC2 用户指南中的使用 SSH 连接您的 Linux 实例。

如果您想在登录后设置 Jupyter Notebook 服务器，请随身携带 SSH 登录命令的副本。要连接到 Jupyter 网页，可以使用该命令的变体。

后续步骤

在 DLAMI 实例上设置 Jupyter Notebook 服务器

在 DLAMI 实例上设置 Jupyter Notebook 服务器

使用 Jupyter Notebook 服务器，可以从 DLAMI 实例创建和运行 Jupyter Notebook。使用 Jupyter 笔记本，您可以进行机器学习 (ML) 实验以进行训练和推理，同时使用 Amazon 基础架构和访问 DLAMI 中内置的软件包。有关 Jupyter Notebook 的更多信息，请参阅 Jupyter 用户文档网站上的 The Jupyter Notebook。

要设置 Jupyter Notebook 服务器，您必须：

- 在 DLAMI 实例上配置 Jupyter Notebook 服务器。
- 配置客户端以连接到 Jupyter Notebook 服务器。我们提供适用于 Windows、macOS 和 Linux 客户端的配置说明。
- 通过登录 Jupyter Notebook 服务器，测试设置。

要完成这些步骤，请遵循以下主题中的说明。设置 Jupyter Notebook 服务器后，您可以运行中附带的示例笔记本教程。DLAMIs有关更多信息，请参阅 运行 Jupyter 笔记本电脑教程。

主题

在 DLAMI 实例上保护 Jupyter Notebook 服务器

为了保护 Jupyter Notebook 服务器的安全，我们建议设置密码并为服务器创建 SSL 证书。要配置密码和 SSL，请先连接到 DLAMI 实例，然后按照以下说明进行操作。

保护 Jupyter Notebook 服务器

1. Jupyter 提供了一个密码实用工具。运行以下命令，在命令提示符处输入您的首选密码。

   ```
   $ jupyter notebook password
   ```

   输出类似如下：

   ```
   Enter password:
   ```

```
    Verify password:
    [NotebookPasswordApp] Wrote hashed password to /home/ubuntu/.jupyter/
jupyter_notebook_config.json
```

2. 创建自签名 SSL 证书。按照提示填写您认为适当的区域。如果要将提示留空，则必须输入 .。您的答案将不会影响证书的功能性。

```
$ cd ~
    $ mkdir ssl
    $ cd ssl
    $ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out
 mycert.pem
```

> **Note**
>
> 您可能希望创建一个常规的 SSL 证书，该证书由第三方签名，且不会导致浏览器向您发出安全警告。此过程涉及内容较多。有关更多信息，请参阅 Jupyter Notebook 用户文档中的 Securing a notebook server。

后续步骤

在 DLAMI 实例上启动 Jupyter Notebook 服务器

在 DLAMI 实例上启动 Jupyter Notebook 服务器

在使用密码和 SSL 保护 Jupyter Notebook 服务器后，可以启动服务器。登录 DLAMI 实例并运行以下命令，该命令使用您之前创建的 SSL 证书。

```
$ jupyter notebook --certfile=~/ssl/mycert.pem --keyfile ~/ssl/mykey.key
```

启动服务器后，即可通过 SSH 隧道从您的客户端计算机连接到服务器。当服务器运行时，您将看到来自 Jupyter 的一些输出，这些输出证实服务器正在运行。此时，请忽略表示您可以通过本地主机 URL 访问服务器的标注，因为在您创建隧道之前，这是行不通的。

> **Note**
>
> 当您使用 Jupyter Web 界面切换框架时，Jupyter 将处理切换环境。有关更多信息，请参阅 通过 Jupyter 切换环境。

后续步骤

在 DLAMI 实例上将客户端连接到 Jupyter Notebook 服务器

在 DLAMI 实例上将客户端连接到 Jupyter Notebook 服务器

在 DLAMI 实例上启动 Jupyter Notebook 服务器后，请将 Windows、macOS 或 Linux 客户端配置为连接到服务器。当您连接时，可以在工作区中的服务器上创建和访问 Jupyter Notebook，并在服务器上运行深度学习代码。

先决条件

请确保您拥有设置 SSH 隧道所需的以下信息：

- 您的 Amazon EC2 实例的公有 DNS 名称。有关更多信息，请参阅亚马逊 EC2 用户指南中的亚马逊 EC2 实例主机名类型。
- 私有密钥文件的密钥对。有关访问密钥对的更多信息，请参阅亚马逊 EC2 用户指南中的亚马逊 EC2 密钥对和亚马逊 EC2 实例。

从 Windows、macOS 或 Linux 客户端进行连接

要从 Windows、macOS 或 Linux 客户端连接到 DLAMI 实例，请按照客户端操作系统的说明进行操作。

Windows

使用 SSH 从 Windows 客户端连接到 DLAMI 实例

1. 使用适用于 Windows 的 SSH 客户端，例如 PuTTY。有关说明，请参阅亚马逊 EC2 用户指南中的使用 PuTTY 连接您的 Linux 实例。有关其它 SSH 连接选项，请参阅使用 SSH 连接到 Linux 实例。
2. （可选）创建通往正在运行的 Jupyter 服务器的 SSH 隧道。在 Windows 客户端上安装 Git Bash，然后按照 macOS 和 Linux 客户端的连接说明进行操作。

macOS or Linux

使用 SSH 从 macOS 或 Linux 客户端连接到 DLAMI 实例

1. 打开终端。

2. 运行以下命令将本地端口 8888 上的所有请求转发到远程 Ama EC2 zon 实例上的端口 8888。通过替换访问亚马逊实例的密钥位置和亚马逊 EC2 实例的公有 DNS 名称来更新命令。 EC2 注意，对于 Amazon Linux AMI，用户名是 `ec2-user` 而非 `ubuntu`。

```
$ ssh -i ~/mykeypair.pem -N -f -L 8888:localhost:8888 ubuntu@ec2-###-##-##-###.compute-1.amazonaws.com
```

此命令在您的客户端和运行 Jupyter Notebook 服务器的远程 Amazon EC2 实例之间打开一条隧道。

后续步骤

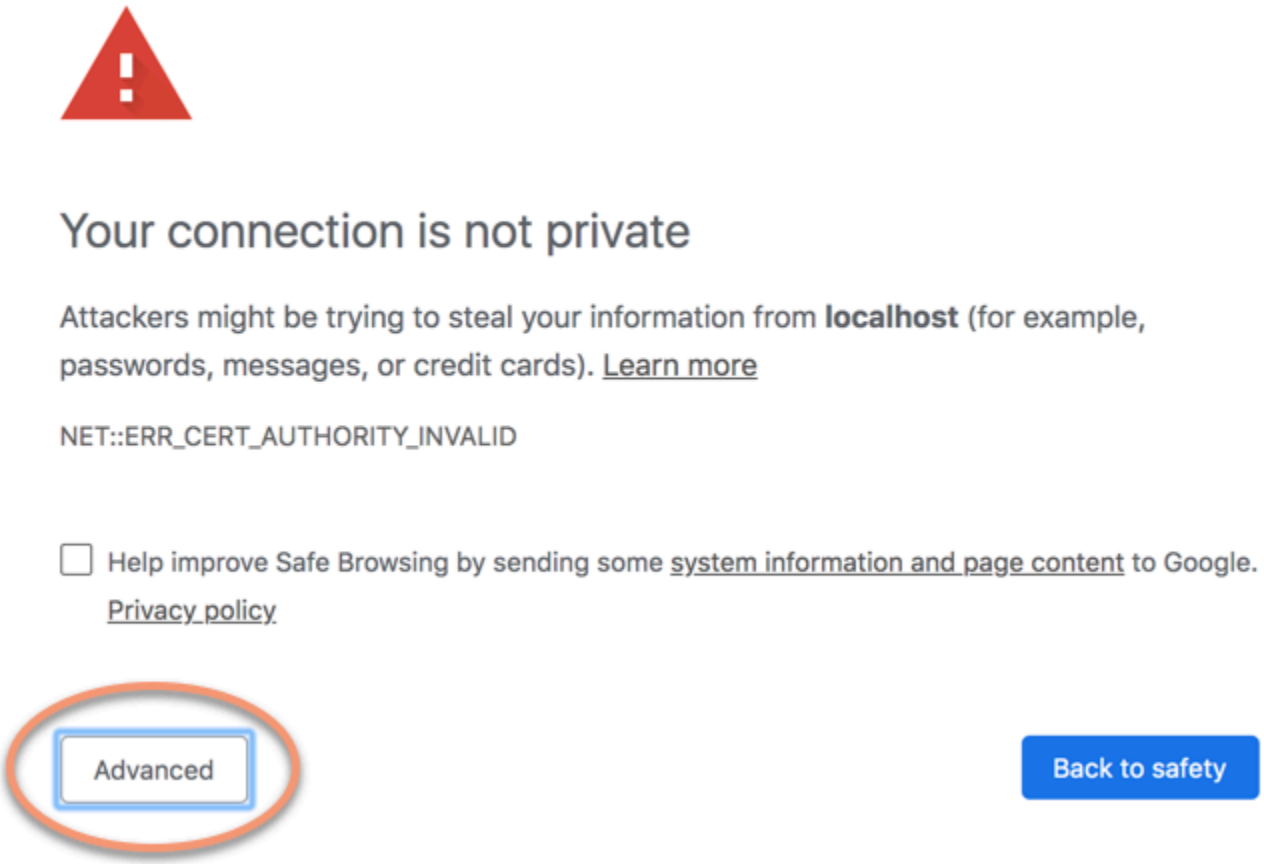
在 DLAMI 实例上登录 Jupyter Notebook 服务器

在 DLAMI 实例上登录 Jupyter Notebook 服务器

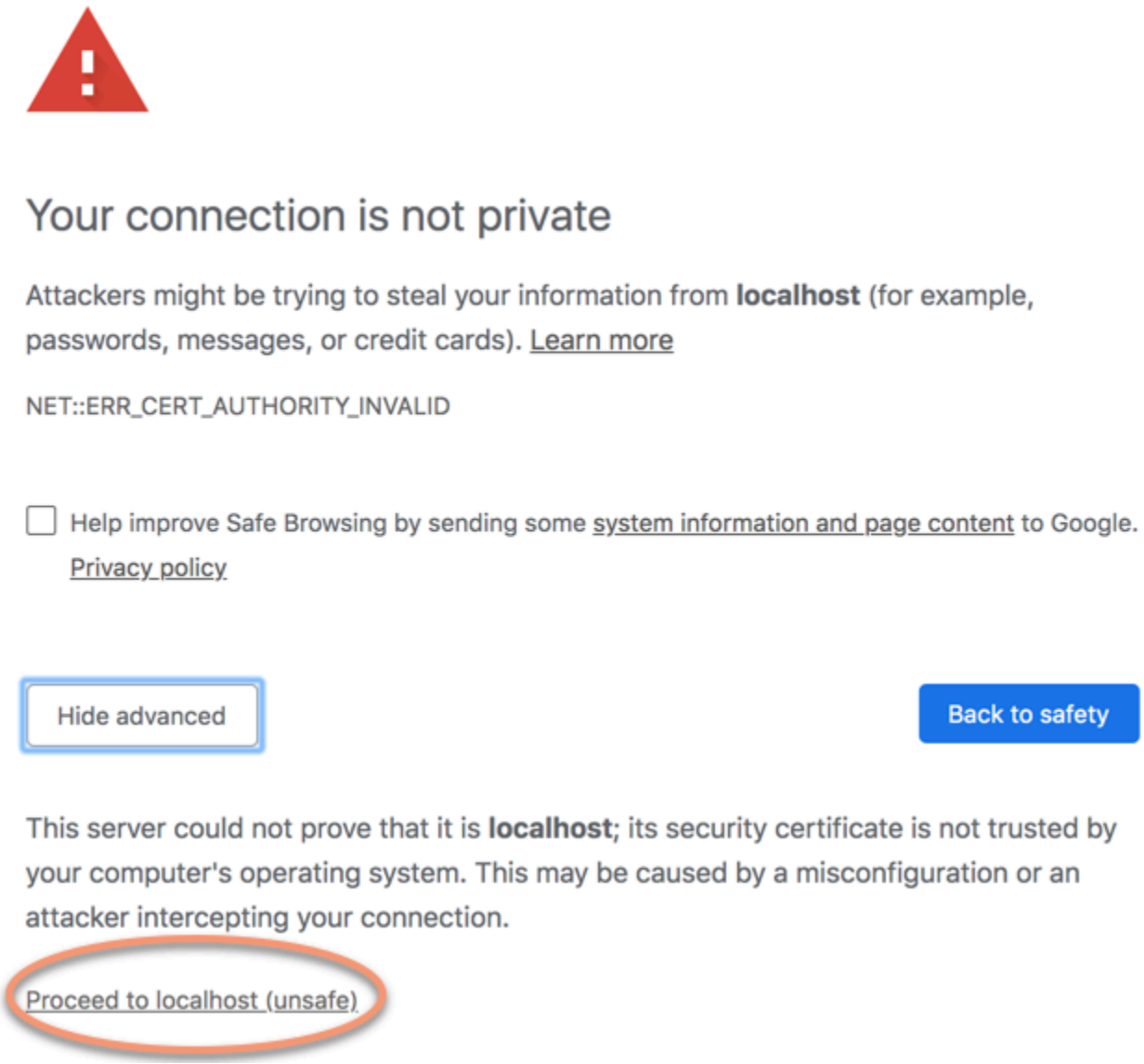
在 DLAMI 实例上将客户端连接到 Jupyter Notebook 服务器后，可以登录服务器。

在浏览器中登录服务器

1. 在浏览器的地址栏中，输入以下 URL，或单击此链接：https://localhost:8888
2. 借助自签名 SSL 证书，浏览器会警告和提示您避免继续访问网站。



由于这是您自己设置的，所以可以安全地继续。根据您的浏览器情况，有可能出现“高级”、“显示详细信息”等类似按钮。



单击此消息，然后单击“前进到本地主机”链接。如果连接成功，则会看到 Jupyter Notebook 服务器网页。此时，系统将要求您提供先前设置的密码。

现在，您可以访问在 DLAMI 实例上运行的 Jupyter Notebook 服务器了。您可以创建新的笔记本或运行提供的教程。

清理 DLAMI 实例

当您不再需要您的 DLAMI 实例时，您可以在 A EC2 mazon 上将其停止或终止，以免产生意外费用。

如果您停止某个实例，可以保留它，稍后当您想再次使用它时再启动它。配置、文件和其它非易失性信息存储在 Amazon Simple Storage Service（Amazon S3）上的卷中。当实例停止时，您会因保留卷而产生 S3 费用，但不会因计算资源而产生费用。当您再次启动实例时，它将装载包含数据的存储卷。

如果您终止实例，实例将消失，您将无法再次启动它。当然，对于终止的实例，您不会再为计算资源支付任何费用。但是，您的数据仍存储在 Amazon S3 上，并且您可能会继续支付 S3 费用。为了防止与终止的实例相关的所有进一步费用，还必须删除 Amazon S3 上的存储卷。有关说明，请参阅亚马逊 EC2 用户指南中的终止亚马逊 EC2 实例。

有关亚马逊 EC2 实例状态（例如`stopped`和）的更多信息`terminated`，请参阅亚马逊 EC2 用户指南中的亚马逊 EC2 实例状态更改。

使用 DLAMI

主题

以下部分介绍如何使用带 Conda 的深度学习 AMI来切换环境、从每个框架运行示例代码以及运行 Jupyter，以便您可以尝试不同的 Notebook 教程。

使用带 Conda 的深度学习 AMI

主题

带 Conda 的深度学习 AMI 的简介

Conda 是一个开源程序包管理系统和环境管理系统，在 Windows、macOS 和 Linux 上运行。Conda 快速安装、运行和更新程序包及其依赖项。Conda 可轻松创建、保存、加载和切换本地计算机上的环境。

带 Conda 的深度学习 AMI 已配置完成，以便让您轻松切换深度学习环境。以下说明为您介绍与 conda 相关的一些基本命令。它们还可以帮助您验证框架的基本导入正常运行，并且您可以使用框架运行一些简单操作。然后，您可以继续查看随 DLAMI 提供的更全面的教程，或者每个框架的项目站点上提供的框架示例。

登录到你的 DLAMI

登录服务器后，您会看到服务器的“每日消息”（MOTD），它介绍了可以用来切换不同深度学习框架的各种 Conda 命令。以下是示例 MOTD。由于新版本 DLAMI 的发布，您的特定 MOTD 可能不同。

```
=============================================================================
        AMI Name: Deep Learning OSS Nvidia Driver AMI (Amazon Linux 2) Version 77
        Supported EC2 instances: G4dn, G5, G6, Gr6, P4d, P4de, P5
            * To activate pre-built tensorflow environment, run: 'source activate
 tensorflow2_p310'
            * To activate pre-built pytorch environment, run: 'source activate
 pytorch_p310'
            * To activate pre-built python3 environment, run: 'source activate python3'

        NVIDIA driver version: 535.161.08

    CUDA versions available: cuda-11.7 cuda-11.8 cuda-12.0 cuda-12.1 cuda-12.2

    Default CUDA version is 12.1

    Release notes: https://docs.aws.amazon.com/dlami/latest/devguide/appendix-ami-
release-notes.html
    AWS Deep Learning AMI Homepage: https://aws.amazon.com/machine-learning/amis/
    Developer Guide and Release Notes: https://docs.aws.amazon.com/dlami/latest/
devguide/what-is-dlami.html
    Support: https://forums.aws.amazon.com/forum.jspa?forumID=263
    For a fully managed experience, check out Amazon SageMaker at https://
aws.amazon.com/sagemaker
    =============================================================================
```

启动 TensorFlow 环境

> **Note**
>
> 在启动您的第一个 Conda 环境时，请在其加载期间耐心等待。带有 Conda 的深度学习 AMI 会在框架首次激活时自动为您的 EC2 实例安装最优化的框架版本。您不应期望后续的延迟。

1. 激活 Python 3 的 TensorFlow 虚拟环境。

```
$ source activate tensorflow2_p310
```

2. 启动 iPython 终端。

```
(tensorflow2_p310)$ ipython
```

3. 运行一个快速 TensorFlow 程序。

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

您应该会看到“Hello, Tensorflow!”

后续步骤

运行 Jupyter 笔记本电脑教程

切换到 PyTorch Python 3 环境

如果您仍然处于 iPython 控制台中，则使用 `quit()`，然后准备切换环境。

- 激活 Python 3 的 PyTorch 虚拟环境。

```
$ source activate pytorch_p310
```

测试一些 PyTorch 代码

要测试您的安装，请使用 Python 编写用于创建和打印数组的 PyTorch 代码。

1. 启动 iPython 终端。

```
(pytorch_p310)$ ipython
```

2. 导入 PyTorch。

```
import torch
```

   您可能会看到一条关于第三方软件包的警告消息。您可以忽略它。

3. 创建一个 5x3 矩阵，将元素随机初始化。打印数组。

```
x = torch.rand(5, 3)
print(x)
```

验证结果。

```
tensor([[0.3105, 0.5983, 0.5410],
        [0.0234, 0.0934, 0.0371],
        [0.9740, 0.1439, 0.3107],
        [0.6461, 0.9035, 0.5715],
        [0.4401, 0.7990, 0.8913]])
```

删除环境

如果您用尽了 DLAMI 上的空间，则可以选择卸载不用的 Conda 软件包：

```
conda env list
conda env remove --name <env_name>
```

使用深度学习基础 AMI

使用深度学习基础 AMI

Base AMI 附带 GPU 驱动程序基础平台，以及可用来部署您自己的自定义深度学习环境的加速库。默认情况下，该 AMI 配置为采用任何一种 NVIDIA CUDA 版本环境。您也可以在不同版本的 CUDA 之间切换。有关如何执行此操作，请参阅以下说明。

配置 CUDA 版本

您可以通过运行 NVIDIA 的 nvcc 程序来验证 CUDA 版本。

```
nvcc --version
```

您可以使用以下 bash 命令来选择并验证特定 CUDA 版本。

```
sudo rm /usr/local/cuda
sudo ln -s /usr/local/cuda-12.0 /usr/local/cuda
```

有关更多信息，请参阅基础 DLAMI 发布说明。

运行 Jupyter 笔记本电脑教程

每个深度学习项目的源代码都附带了教程和示例，大多数情况下，它们可以在任何 DLAMI 上运行。如果您选择了 带 Conda 的深度学习 AMI，那么您将获得一些已经建立并准备好尝试的精选教程的额外好处。

> **Important**
>
> 要运行安装在 DLAMI 上的 Jupyter 笔记本教程，您需要 在 DLAMI 实例上设置 Jupyter Notebook 服务器。

Jupyter 服务器运行后，您可以通过 Web 浏览器运行这些教程。如果您正在运行带 Conda 的深度学习 AMI，或者如果您已经建立了 Python 环境，则可以从 Jupyter 笔记本界面切换 Python 内核。在尝试运行特定于框架的教程之前，请选择合适的内核。我们为带 Conda 的深度学习 AMI 的用户提供了更多这方面的示例。

> **Note**
>
> 许多教程都需要额外 Python 模块，您的 DLAMI 上可能尚未设置这些模块。如果您收到类似 `"xyz module not found"` 的错误，请登录到 DLAMI，激活环境（如上所述），然后安装必要的模块。

> **Tip**
>
> 深度学习教程和示例通常依赖于一个或多个 GPUs。如果您的实例类型没有 GPU，您可能需要更改一些示例代码才能使其运行。

导航已安装的教程

一旦登录到 Jupyter 服务器且可以看到教程目录（仅限带 Conda 的深度学习 AMI 上）时，就会看到按每个框架名称排列的教程文件夹。如果您没有看到某个框架，则表明在您当前 DLAMI 上该框架的教程不可用。单击框架名称以查看列出的教程，然后单击一个教程，将其启动。

在带 Conda 的深度学习 AMI 上第一次运行 Notebook 时，它会想知道您要使用哪个环境。它会提示您从列表中进行选择。每个环境都根据以下模式命名：

`Environment (conda_framework_python-version)`

例如，你可能会看到`Environment (conda_mxnet_p36)`，这表示环境中有 MXNet Python 3。另一个变体是`Environment (conda_mxnet_p27)`，这表示环境中有 MXNet Python 2。

> **Tip**
>
> 如果您想知道哪个版本的 CUDA 处于活动状态，一种查看方法是在首次登录到 DLAMI 时在 MOTD 中查看。

通过 Jupyter 切换环境

如果您决定尝试一个不同框架的教程，一定要验证当前正在运行的内核。此信息可以在 Jupyter 界面的右上方看到，就在注销按钮的下方。您可以在任何打开的笔记本电脑上更改内核，方法是依次单击 Kernel、Change Kernel 菜单项，然后单击正运行的笔记本电脑适合的环境。

此时您需要重新运行任何单元，因为内核中的更改将会擦除之前运行的任何内容的状态。

> **Tip**
>
> 在框架之间进行切换可能很有趣而且很有教育意义，但是可能导致耗尽内存。如果您开始遇到错误，请查看运行 Jupyter 服务器的终端窗口。这里有一些有用的消息和错误记录，你可能会看到一个 out-of-memory错误。要解决这一问题，您可以转到 Jupyter 服务器的主页中，单击 Running 选项卡，然后为每个教程单击 Shutdown，因为这些教程可能仍然在后台运行，导致耗尽了所有内存。

教程

以下是有关如何使用带 Conda 的深度学习 AMI 的软件的教程。

主题

- 激活框架
- 使用 Elastic Fabric Adapter 进行分布式训练
- GPU 监控和优化

- 带有 DLAMI 的 Amazon 推理芯片
- ARM64 DLAMI
- 推理
- 模型处理

激活框架

以下是在带 Conda 的深度学习 AMI 上安装的深度学习框架。单击某个框架即可了解如何将其激活。

主题

- PyTorch
- TensorFlow 2

PyTorch

正在激活 PyTorch

当框架的稳定 Conda 程序包发布时，它会在 DLAMI 上进行测试并预安装。如果您希望运行最新的、未经测试的每日构建版本，您可以手动Install PyTorch 的夜间构建（实验版）。

要激活当前安装的框架，请按照这些有关带 Conda 的深度学习 AMI 的说明进行操作。

对于使用 PyTorch CUDA 和 MKL-DNN 的 Python 3，请运行以下命令：

```
$ source activate pytorch_p310
```

启动 iPython 终端。

```
(pytorch_p310)$ ipython
```

运行一个快速 PyTorch 程序。

```
import torch
x = torch.rand(5, 3)
print(x)
print(x.size())
y = torch.rand(5, 3)
print(torch.add(x, y))
```

您应该会看到系统输出初始随机数组，然后输出大小，然后添加另一个随机数组。

Install PyTorch 的夜间构建（实验版）

如何 PyTorch 从夜间版本中安装

您可以使用 Conda 将最新 PyTorch 版本安装到深度学习 AMI 上的任一或两个 PyTorch Conda 环境中。

1. • （Python 3 的选项）-激活 Python 3 PyTorch 环境：

```
$ source activate pytorch_p310
```

2. 其余步骤假定您使用的是 `pytorch_p310` 环境。移除当前安装的 PyTorch：

```
(pytorch_p310)$ pip uninstall torch
```

3. • （GPU 实例的选项）-使用 CUDA.0 安装最新的夜间版本： PyTorch

```
(pytorch_p310)$ pip install torch_nightly -f https://download.pytorch.org/whl/
nightly/cu100/torch_nightly.html
```

• （CPU 实例的选项）-为不 GPUs带以下选项的 PyTorch 实例安装最新的夜间版本：

```
(pytorch_p310)$ pip install torch_nightly -f https://download.pytorch.org/whl/
nightly/cpu/torch_nightly.html
```

4. 要验证您是否已成功安装最新的夜间版本，请启动 IPython 终端并检查的版本。 PyTorch

```
(pytorch_p310)$ ipython
```

```
import torch
print (torch.__version__)
```

输出应类似于以下内容：`1.0.0.dev20180922`

5. 要验证 PyTorch 夜间版本是否与 MNIST 示例配合使用，您可以从 PyTorch的示例存储库中运行测试脚本：

```
(pytorch_p310)$ cd ~
(pytorch_p310)$ git clone https://github.com/pytorch/examples.git pytorch_examples
(pytorch_p310)$ cd pytorch_examples/mnist
```

```
(pytorch_p310)$ python main.py || exit 1
```

更多教程

有关更多教程和示例，请参阅该框架的官方PyTorch 文档、文档和PyTorch网站。

TensorFlow 2

本教程介绍如何在运行带有 Conda 的深度学习 AMI（Conda 上的 DLAMI）的实例上激活 TensorFlow 2 并运行 2 程序。 TensorFlow

当框架的稳定 Conda 程序包发布时，它会在 DLAMI 上进行测试并预安装。

正在激活 TensorFlow 2

和 Cond TensorFlow a 一起在 DLAMI 上跑步

1. 要激活 TensorFlow 2，请使用 Conda 打开 DLAMI 的亚马逊弹性计算云 (亚马逊 EC2) 实例。
2. 对于使用 TensorFlow CUDA 10.1 和 MKL-DNN 的 Python 3 上的 2 和 Keras 2，请运行以下命令：

   ```
   $ source activate tensorflow2_p310
   ```
3. 启动 iPython 终端：

   ```
   (tensorflow2_p310)$ ipython
   ```
4. 运行 TensorFlow 2 程序以验证它是否正常运行：

   ```
   import tensorflow as tf
   hello = tf.constant('Hello, TensorFlow!')
   tf.print(hello)
   ```

   `Hello, TensorFlow!` 应显示在您的屏幕上。

更多教程

有关更多教程和示例，请参阅 TensorFlow Python API 的 TensorFlow 文档或TensorFlow访问网站。

使用 Elastic Fabric Adapter 进行分布式训练

Elastic Fabric Adapter (EFA) 是一种网络设备，可以将其附加到您的 DLAMI 实例以加快高性能计算 (HPC) 应用程序的速度。借助 Amazon 云提供的可扩展性、灵活性和弹性，EFA 使您能够实现本地 HPC 集群的应用程序性能。

以下主题将向您展示如何开始结合使用 EFA 与 DLAMI。

> **Note**
>
> 从这个基础 GPU DLAMI 列表中选择您的 DLAMI

主题

- 使用 EFA 启动 Amazon Deep Learning AMIs 实例
- 在 DLAMI 上使用 EFA

使用 EFA 启动 Amazon Deep Learning AMIs 实例

最新基础 DLAMI 可随时与 EFA 结合使用，并随附所需的驱动程序、内核模块、libfabric、openmpi 和适用于 GPU 实例的 NCCL OFI 插件。

您可以在发布说明中找到基础 DLAMI 的支持 CUDA 版本。

注意：

- 在 EFA 上使用 `mpirun` 运行 NCCL 应用程序时，必须将 EFA 支持的安装的完整路径指定为：

```
/opt/amazon/openmpi/bin/mpirun <command>
```

- 要使您的应用程序能够使用 EFA，请将 `FI_PROVIDER="efa"` 添加到 `mpirun` 命令，如在 DLAMI 上使用 EFA中所示。

主题

- 准备 EFA 启用的安全组
- 启动实例
- 验证 EFA 附件

准备 EFA 启用的安全组

EFA 需要一个安全组来支持进出安全组本身的所有入站和出站流量。有关更多信息，请参阅 EFA 文档。

1. 打开亚马逊 EC2 控制台，网址为https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择安全组，然后选择创建安全组。
3. 在创建安全组窗口中，执行以下操作：
   - 对于安全组名称，请输入一个描述性的安全组名称，例如 `EFA-enabled security group`。
   - （可选）对于描述，请输入安全组的简要描述。
   - 对于 VPC，请选择要在其中启动启用了 EFA 的实例的 VPC。
   - 选择创建。
4. 选择您创建的安全组，然后在描述 选项卡上复制组 ID。
5. 在入站和出站选项卡上，执行以下操作：
   - 选择编辑。
   - 对于类型，请选择所有流量。
   - 对于 Source，选择 Custom。
   - 将您复制的安全组 ID 粘贴到该字段中。
   - 选择保存。
6. 启用入站流量，请参考授权您 Linux 实例的入站流量。如果跳过此步骤，您将无法与您的 DLAMI 实例进行通信。

启动实例

目前 Amazon Deep Learning AMIs ，以下实例类型和操作系统支持上的 EFA：

- p3dn：亚马逊 Linux 2、Ubuntu 20.04
- p4d、p4de：亚马逊 Linux 2、亚马逊 Linux 2023、Ubuntu 20.04、Ubuntu 22.04
- P5、p5e、p5eN：亚马逊 Linux 2、亚马逊 Linux 2023、Ubuntu 20.04、Ubuntu 22.04、Ubuntu 22.04

以下部分介绍了如何启动 EFA 启用的 DLAMI 实例。有关启动 EFA 启用的 DLAMI 实例的更多信息，请参阅在集群置放群组中启动 EFA 启用的实例。

1. 打开亚马逊 EC2 控制台，网址为https://console.aws.amazon.com/ec2/。
2. 选择启动实例。
3. 在选择 AMI 页面上，选择在 DLAMI Release Notes Page 上找到的受支持的 DLAMI。
4. 在选择实例类型页面上，选择以下支持的实例类型之一，然后选择下一步：配置实例详细信息。有关受支持的实例的列表，请参阅此链接：开始使用 EFA 和 MPI。
5. 在配置实例详细信息页面中，执行以下操作：
   - 对于实例的数量，请输入要启动的启用了 EFA 的实例数量。
   - 对于网络 和子网，请选择要在其中启动实例的 VPC 和子网。
   - [可选] 对于置放群组，请选择将实例添加到置放群组。为获得最佳性能，请在置放群组中启动实例。
   - [可选] 对于置放群组名称，请选择添加到新的置放群组，输入置放群组的描述性名称，然后对于置放群组策略选择集群。
   - 请务必在此页面上启用“Elastic Fabric Adapter”。如果禁用此选项，请将子网更改为支持所选实例类型的子网。
   - 在网络接口部分中，为设备 eth0 选择新网络接口。您可以选择指定一个主 IPv4 地址和一个或多个辅助 IPv4 地址。如果您要在具有关联 IPv6 CIDR 块的子网中启动实例，则可以选择指定一个主 IPv6 地址和一个或多个辅助 IPv6 地址。
   - 选择下一步：添加存储。
6. 在添加存储页面上，除了 AMI 指定的卷（如根设备卷）以外，还要指定要附加到实例的卷，然后选择下一步：添加标签。
7. 在添加标签页面上，为实例指定标签（例如，便于用户识别的名称），然后选择下一步：配置安全组。
8. 在配置安全组页面上，对于分配安全组选择选择一个现有的安全组，然后选择先前创建的安全组。
9. 选择审核并启动。
10. 在核查实例启动页面上，检查这些设置，然后选择启动以选择一个密钥对并启动您的实例。

验证 EFA 附件

通过控制台

启动实例后，请在 Amazon 控制台中查看实例详细信息。为此，请在 EC2 控制台中选择实例，然后查看页面下方窗格中的描述选项卡。找到参数“Network Interfaces: eth0”，然后单击 eth0，这将弹出一个弹出窗口。确保已启用“Elastic Fabric Adapter”。

如果未启用 EFA，您可以通过以下任一方式解决此问题：

- 使用相同的步骤终止 EC2 实例并启动一个新实例。确保已附加 EFA。
- 将 EFA 附加到现有实例。

1. 在 EC2 控制台中，转到网络接口。
2. 单击“创建虚拟网络接口”。
3. 选择您的实例所在的相同子网。
4. 确保启用“Elastic Fabric Adapter”并点击“创建”。
5. 返回 EC2 实例选项卡并选择您的实例。
6. 转到“操作：实例状态”并在附加 EFA 之前停止实例。
7. 从 “操作”中，选择“网络连接：连接网络接口”。
8. 选择您刚刚创建的界面，然后点击“附加”。
9. 重新启动您的实例。

通过实例

以下测试脚本已存在于 DLAMI 中。运行它以确保内核模块正确加载。

```
$ fi_info -p efa
```

您的输出应类似于以下内容。

```
provider: efa
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-rdm
    version: 2.0
    type: FI_EP_RDM
    protocol: FI_PROTO_EFA
provider: efa
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-dgrm
    version: 2.0
    type: FI_EP_DGRAM
    protocol: FI_PROTO_EFA
provider: efa;ofi_rxd
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-dgrm
```

```
    version: 1.0
    type: FI_EP_RDM
    protocol: FI_PROTO_RXD
```

验证安全组配置

以下测试脚本已存在于 DLAMI 中。运行它以确保您创建的安全组配置正确。

```
$ cd /opt/amazon/efa/test/
$ ./efa_test.sh
```

您的输出应类似于以下内容。

```
Starting server...
Starting client...
bytes   #sent   #ack     total       time     MB/sec    usec/xfer   Mxfers/sec
64      10      =10      1.2k        0.02s      0.06    1123.55       0.00
256     10      =10      5k          0.00s     17.66      14.50       0.07
1k      10      =10      20k         0.00s     67.81      15.10       0.07
4k      10      =10      80k         0.00s    237.45      17.25       0.06
64k     10      =10      1.2m        0.00s    921.10      71.15       0.01
1m      10      =10      20m         0.01s   2122.41     494.05       0.00
```

如果它停止响应或未完成，请确保您的安全组具有正确的入站/出站规则。

在 DLAMI 上使用 EFA

以下部分描述如何在 Amazon Deep Learning AMIs上使用 EFA 来运行多节点应用程序。

使用 EFA 来运行多节点应用程序

要跨节点集群运行应用程序，需要以下配置

主题

- 启用无密码 SSH
- 创建主机文件
- NCCL 测试

启用无密码 SSH

选择集群中的一个节点作为领导节点。其余节点称为成员节点。

1. 在领导节点上，生成 RSA 密钥对。

```
ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
```

2. 更改领导节点上私有密钥的权限。

```
chmod 600 ~/.ssh/id_rsa
```

3. 复制公有密钥 ~/.ssh/id_rsa.pub 并将其附加到集群中成员节点的 ~/.ssh/authorized_keys 之后。
4. 现在，您应该能够使用私有 ip 从领导节点直接登录到成员节点。

```
ssh <member private ip>
```

5. 通过将以下内容添加到领导节点上的 ~/.ssh/config 文件中，禁用 strictHostKey检查并在领导节点上启用代理转发：

```
Host *
    ForwardAgent yes
Host *
    StrictHostKeyChecking no
```

6. 在 Amazon Linux 2 实例上，在领导节点上运行以下命令，为配置文件提供正确的权限：

```
chmod 600 ~/.ssh/config
```

创建主机文件

在领导节点上，创建主机文件以标识集群中的节点。主机文件必须针对集群中的每个节点都有一个条目。创建文件 ~/hosts 并使用私有 IP 添加每个节点，如下所示：

```
localhost slots=8
<private ip of node 1> slots=8
<private ip of node 2> slots=8
```

NCCL 测试

> **Note**
>
> 这些测试是使用 EFA 版本 1.38.0 和 OFI NCCL Plugin 1.13.2 运行的。

下面列出了由 Nvidia 提供的 NCCL 测试子集，用于测试多个计算节点的功能和性能

支持的实例：p3dn、P4、P5、p5e、p5en

性能测试

P4d.24xlarge 上的多节点 NCCL 性能测试

要使用 EFA 来检查 NCCL 性能，请运行官方 NCCL-Tests 存储库中提供的标准 NCCL 性能测试。DLAMI 附带了这个已经为 CUDA XX.X 构建的测试。您也可以使用 EFA 运行自己的脚本。

构建您自己的脚本时，请参阅以下指南：

- 当使用 EFA 来运行 NCCL 应用程序时，按照示例所示使用到 mpirun 的完整路径。
- 根据集群 GPUs 中的实例数量更改参数 np 和 N。
- 添加 NCCL_DEBUG=INFO 标志，并确保日志将 EFA 用法指示为“所选提供程序是 EFA”。
- 设置要解析的训练日志位置以进行验证

```
TRAINING_LOG="testEFA_$(date +"%N").log"
```

在任何成员节点上使用 `watch nvidia-smi` 命令来监视 GPU 使用情况。以下 `watch nvidia-smi` 命令适用于通用 CUDA xx.x 版本，并且依赖于您的实例的操作系统。您可以通过替换脚本中的 CUDA 版本来运行适用于您的 Amazon EC2 实例中任何可用的 CUDA 版本的命令。

- 亚马逊 Linux 2、亚马逊 Linux 2023：

```
 $ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \
-x NCCL_DEBUG=INFO --mca pml ^cm \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:/opt/amazon/efa/lib64:/opt/amazon/openmpi/
lib64:$LD_LIBRARY_PATH \
--hostfile hosts --mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to
 none \
```

```
/usr/local/cuda-xx.x/efa/test-cuda-xx.x/all_reduce_perf -b 8 -e 1G -f 2 -g 1 -c 1 -n
 100 | tee ${TRAINING_LOG}
```

- Ubuntu 20.04，Ubuntu 20.04：

```
$ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \
-x NCCL_DEBUG=INFO --mca pml ^cm \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:/opt/amazon/efa/lib:/opt/amazon/openmpi/
lib:$LD_LIBRARY_PATH \
--hostfile hosts --mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to
 none \
/usr/local/cuda-xx.x/efa/test-cuda-xx.x/all_reduce_perf -b 8 -e 1G -f 2 -g 1 -c 1 -n
 100 | tee ${TRAINING_LOG}
```

您的输出应与以下内容类似：

```
# nThread 1 nGpus 1 minBytes 8 maxBytes 1073741824 step: 2(factor) warmup iters: 5
 iters: 100 agg iters: 1 validation: 1 graph: 0
#
# Using devices
#  Rank  0 Group  0 Pid  33378 on ip-172-31-42-25 device  0 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  1 Group  0 Pid  33379 on ip-172-31-42-25 device  1 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  2 Group  0 Pid  33380 on ip-172-31-42-25 device  2 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank  3 Group  0 Pid  33381 on ip-172-31-42-25 device  3 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank  4 Group  0 Pid  33382 on ip-172-31-42-25 device  4 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank  5 Group  0 Pid  33383 on ip-172-31-42-25 device  5 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank  6 Group  0 Pid  33384 on ip-172-31-42-25 device  6 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank  7 Group  0 Pid  33385 on ip-172-31-42-25 device  7 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank  8 Group  0 Pid  30378 on ip-172-31-43-8 device  0 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  9 Group  0 Pid  30379 on ip-172-31-43-8 device  1 [0x10] NVIDIA A100-SXM4-40GB
#  Rank 10 Group  0 Pid  30380 on ip-172-31-43-8 device  2 [0x20] NVIDIA A100-SXM4-40GB
#  Rank 11 Group  0 Pid  30381 on ip-172-31-43-8 device  3 [0x20] NVIDIA A100-SXM4-40GB
#  Rank 12 Group  0 Pid  30382 on ip-172-31-43-8 device  4 [0x90] NVIDIA A100-SXM4-40GB
```

```
#  Rank 13 Group  0 Pid  30383 on ip-172-31-43-8 device  5 [0x90] NVIDIA A100-SXM4-40GB
#  Rank 14 Group  0 Pid  30384 on ip-172-31-43-8 device  6 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank 15 Group  0 Pid  30385 on ip-172-31-43-8 device  7 [0xa0] NVIDIA A100-SXM4-40GB
ip-172-31-42-25:33385:33385 [7] NCCL INFO cudaDriverVersion 12060
ip-172-31-43-8:30383:30383 [5] NCCL INFO Bootstrap : Using ens32:172.31.43.8
ip-172-31-43-8:30383:30383 [5] NCCL INFO NCCL version 2.23.4+cuda12.5
...
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.13.2-aws
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Using Libfabric version 1.22
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Using CUDA driver version 12060 with
 runtime 12050
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting NCCL_NVLSTREE_MAX_CHUNKSIZE
 to 512KiB
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting NCCL_NVLS_CHUNKSIZE to 512KiB
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Running on p4d.24xlarge platform,
 Setting NCCL_TOPO_FILE environment variable to /opt/amazon/ofi-nccl/share/aws-ofi-
nccl/xml/p4d-24xl-topo.xml
...
------------------------------some output truncated-----------------------------------
#                                                              out-of-place
           in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong
  time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)
  (us)  (GB/s)  (GB/s)
           8             2     float     sum      -1    180.3    0.00    0.00      0
 179.3    0.00    0.00      0
          16             4     float     sum      -1    178.1    0.00    0.00      0
 177.6    0.00    0.00      0
          32             8     float     sum      -1    178.5    0.00    0.00      0
 177.9    0.00    0.00      0
          64            16     float     sum      -1    178.8    0.00    0.00      0
 178.7    0.00    0.00      0
         128            32     float     sum      -1    178.2    0.00    0.00      0
 177.8    0.00    0.00      0
         256            64     float     sum      -1    178.6    0.00    0.00      0
 178.8    0.00    0.00      0
         512           128     float     sum      -1    177.2    0.00    0.01      0
 177.1    0.00    0.01      0
```

```
        1024           256     float     sum      -1    179.2    0.01    0.01       0
 179.3    0.01    0.01      0
        2048           512     float     sum      -1    181.3    0.01    0.02       0
 181.2    0.01    0.02      0
        4096          1024     float     sum      -1    184.2    0.02    0.04       0
 183.9    0.02    0.04      0
        8192          2048     float     sum      -1    191.2    0.04    0.08       0
 190.6    0.04    0.08      0
       16384          4096     float     sum      -1    202.5    0.08    0.15       0
 202.3    0.08    0.15      0
       32768          8192     float     sum      -1    233.0    0.14    0.26       0
 232.1    0.14    0.26      0
       65536         16384     float     sum      -1    238.6    0.27    0.51       0
 235.1    0.28    0.52      0
      131072         32768     float     sum      -1    237.2    0.55    1.04       0
 236.8    0.55    1.04      0
      262144         65536     float     sum      -1    248.3    1.06    1.98       0
 247.0    1.06    1.99      0
      524288        131072     float     sum      -1    309.2    1.70    3.18       0
 307.7    1.70    3.20      0
     1048576        262144     float     sum      -1    408.7    2.57    4.81       0
 404.3    2.59    4.86      0
     2097152        524288     float     sum      -1    613.5    3.42    6.41       0
 607.9    3.45    6.47      0
     4194304       1048576     float     sum      -1    924.5    4.54    8.51       0
 914.8    4.58    8.60      0
     8388608       2097152     float     sum      -1   1059.5    7.92   14.85       0
 1054.3    7.96   14.92      0
    16777216       4194304     float     sum      -1   1269.9   13.21   24.77       0
 1272.0   13.19   24.73      0
    33554432       8388608     float     sum      -1   1642.7   20.43   38.30       0
 1636.7   20.50   38.44      0
    67108864      16777216     float     sum      -1   2446.7   27.43   51.43       0
 2445.8   27.44   51.45      0
   134217728      33554432     float     sum      -1   4143.6   32.39   60.73       0
 4142.4   32.40   60.75      0
   268435456      67108864     float     sum      -1   7351.9   36.51   68.46       0
 7346.7   36.54   68.51      0
   536870912     134217728     float     sum      -1    13717   39.14   73.39       0
 13703   39.18   73.46      0
  1073741824     268435456     float     sum      -1    26416   40.65   76.21       0
 26420   40.64   76.20      0
...
# Out of bounds values : 0 OK
```

```
# Avg bus bandwidth    : 15.5514
```

验证测试

要验证 EFA 测试返回的结果有效，请使用以下测试进行确认：

- 使用实例元数据获取 EC2 实例类型：

```
TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
INSTANCE_TYPE=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" -v http://169.254.169.254/latest/meta-data/instance-type)
```

- 运行性能测试
- 设置以下参数

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- 验证结果，如下所示：

```
RETURN_VAL=`echo $?`
if [ ${RETURN_VAL} -eq 0 ]; then

    # [0] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.13.2-aws
    # [0] NCCL INFO NET/OFI Using CUDA driver version 12060 with runtime 12010

    # cudaDriverVersion 12060  --> This is max supported cuda version by nvidia driver
    # NCCL version 2.23.4+cuda12.5 --> This is NCCL version compiled with cuda version

    # Validation of logs
    grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-specific options text not found"; exit 1; }
    grep "busbw" ${TRAINING_LOG} || { echo "busbw text not found"; exit 1; }
    grep "Avg bus bandwidth " ${TRAINING_LOG} || { echo "Avg bus bandwidth text not found"; exit 1; }
    grep "NCCL version $NCCL_VERSION" ${TRAINING_LOG} || { echo "Text not found: NCCL version $NCCL_VERSION"; exit 1; }
    if [[ ${INSTANCE_TYPE} == "p4d.24xlarge" ]]; then
```

```
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found: NET/
Libfabric/0/GDRDMA"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5e.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5en.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 16 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p3dn.24xlarge" ]]; then
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
    fi
    echo "***************************** check_efa_nccl_all_reduce passed for cuda
 version ${CUDA_VERSION} *****************************"
else
    echo "***************************** check_efa_nccl_all_reduce failed for cuda
 version ${CUDA_VERSION} *****************************"
fi
```

- 要访问基准数据，可以解析多节点 all_reduce 测试的表输出的最后一行：

```
benchmark=$(sudo cat ${TRAINING_LOG} | grep '1073741824' | tail -n1 | awk -F " "
 '{{print $12}}' | sed 's/ //' | sed  's/  5e-07//')
if [[ -z "${benchmark}" ]]; then
  echo "benchmark variable is empty"
  exit 1
fi
```

```
echo "Benchmark throughput: ${benchmark}"
```

GPU 监控和优化

以下部分将指导您完成 GPU 优化和监控选项。本部分的组织方式如同一个典型工作流程一样，其中包含监控监督预处理和训练。

- 监控
  - GPUs 使用监视器 CloudWatch
- 优化
  - 预处理
  - 训练

监控

您的 DLAMI 已预安装多个 GPU 监控工具。本指南还将介绍可用于下载和安装的工具。

- GPUs 使用监视器 CloudWatch-预装的实用程序，可向 Amazon CloudWatch 报告 GPU 使用情况统计信息。
- nvidia-smi CLI — 一个监控总体 GPU 计算和内存利用率的实用工具。它已预先安装在你的 Amazon Deep Learning AMIs (DLAMI) 上。
- NVML C 库 - 一个基于 C 的 API，可直接访问 GPU 监控和管理功能。此项已在后台由 nvidia-smi CLI 所使用且已预安装在您的 DLAMI 上。它还具有 Python 和 Perl 绑定以方便采用这些请求进行开发。你的 DLAMI 上预装的 gpumon.py 实用程序使用的是来自的 pynvml 包。nvidia-ml-py
- NVIDIA DCGM - 一个集群管理工具。请访问开发人员页面，了解如何安装和配置此工具。

> **Tip**
>
> 请查看 NVIDIA 的开发人员博客，了解有关使用已安装在您的 DLAMI 上的 CUDA 工具的最新信息。
>
> - 使用 Nsight IDE 和 nvprof 监控 TensorCore 利用率。

GPUs 使用监视器 CloudWatch

当您将 DLAMI 与 GPU 结合使用时，您可能会发现，您要寻找在训练或推理期间跟踪其使用率的方式。这对于优化您的数据管道以及调整深度学习网络非常有用。

有两种方法可以配置 GPU 指标 CloudWatch：

- 使用 Amazon CloudWatch 代理配置指标（推荐）
- 使用预安装 gpumon.py 脚本来配置指标

使用 Amazon CloudWatch 代理配置指标（推荐）

将您的 DLAMI 与 CloudWatch 统一代理集成，以配置 GPU 指标并监控 Amazon 加速实例中 GPU 协进程的利用率。 EC2

使用 DLAMI 来配置 GPU 指标的方式有四种：

- 配置最低 GPU 指标
- 配置部分 GPU 指标
- 配置所有可用 GPU 指标
- 配置自定义 GPU 指标

有关更新和安全补丁的信息，请参阅 代理的安全补丁 Amazon CloudWatch

先决条件

首先，您必须配置 Amazon EC2 实例 IAM 权限，以允许您的实例将指标推送到 CloudWatch。有关详细步骤，请参阅创建用于 CloudWatch 代理的 IAM 角色和用户。

配置最低 GPU 指标

使用 `dlami-cloudwatch-agent@minimal` `systemd` 服务来配置最低 GPU 指标。此服务配置以下指标：

- `utilization_gpu`
- `utilization_memory`

您可以在以下位置找到适用于最低预先配置 GPU 指标的 `systemd` 服务：

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-minimal.json
```

使用以下命令启用并启动 systemd 服务：

```
sudo systemctl enable dlami-cloudwatch-agent@minimal
sudo systemctl start dlami-cloudwatch-agent@minimal
```

配置部分 GPU 指标

使用 dlami-cloudwatch-agent@partial systemd 服务来配置部分 GPU 指标。此服务配置以下指标：

- utilization_gpu
- utilization_memory
- memory_total
- memory_used
- memory_free

您可以在以下位置找到适用于部分预先配置 GPU 指标的 systemd 服务：

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-partial.json
```

使用以下命令启用并启动 systemd 服务：

```
sudo systemctl enable dlami-cloudwatch-agent@partial
sudo systemctl start dlami-cloudwatch-agent@partial
```

配置所有可用 GPU 指标

使用 dlami-cloudwatch-agent@all systemd 服务来配置所有可用 GPU 指标。此服务配置以下指标：

- utilization_gpu
- utilization_memory
- memory_total
- memory_used

- memory_free
- temperature_gpu
- power_draw
- fan_speed
- pcie_link_gen_current
- pcie_link_width_current
- encoder_stats_session_count
- encoder_stats_average_fps
- encoder_stats_average_latency
- clocks_current_graphics
- clocks_current_sm
- clocks_current_memory
- clocks_current_video

您可以在以下位置找到适用于所有可用预先配置 GPU 指标的 systemd 服务：

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-all.json
```

使用以下命令启用并启动 systemd 服务：

```
sudo systemctl enable dlami-cloudwatch-agent@all
sudo systemctl start dlami-cloudwatch-agent@all
```

配置自定义 GPU 指标

如果预配置的指标不符合您的要求，则可以创建自定义 CloudWatch 代理配置文件。

创建自定义配置文件

要创建自定义配置文件，请参阅手动创建或编辑 CloudWatch 代理配置文件中的详细步骤。

在此示例中，假设架构定义位于 /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json。

使用您的自定义文件来配置指标

运行以下命令根据您的自定义文件配置 CloudWatch 代理：

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config -m ec2 -s -c \
file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
```

代理的安全补丁 Amazon CloudWatch

新发布 DLAMIs 的服务器配置了最新的可用 Amazon CloudWatch 代理安全补丁。请参阅以下部分，根据您选择的操作系统，使用最新安全补丁来更新您当前的DLAMI。

Amazon Linux 2

yum用于获取亚马逊 Linux 2 DLAMI 的最新 Amazon CloudWatch 代理安全补丁。

```
 sudo yum update
```

Ubuntu

要使用Ubuntu获取DLAMI的最新 Amazon CloudWatch 安全补丁，必须 Amazon CloudWatch 使用 Amazon S3下载链接重新安装代理。

```
wget https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/
amazon-cloudwatch-agent.deb
```

有关使用 Amazon S3 下载链接安装 Amazon CloudWatch 代理的更多信息，请参阅在服务器上安装和运行 CloudWatch 代理。

使用预安装 **gpumon.py** 脚本来配置指标

一个名为 gpumon.py 的实用工具已预安装在您的 DLAMI 上。它集成 CloudWatch 并支持监控每个 GPU 的使用情况：GPU 内存、GPU 温度和 GPU 功率。该脚本会定期将监控的数据发送到 CloudWatch。您可以通过更改脚本中的一些设置来配置要发送到 CloudWatch 的数据的粒度级别。但是，在启动脚本之前，您需要进行设置 CloudWatch 才能接收指标。

如何使用设置和运行 GPU 监控 CloudWatch

1. 创建 IAM 用户，或修改现有用户以制定向其发布指标的策略 CloudWatch。如果创建新用户，请记下凭证，因为您将在下一步中需要这些凭证。

   要搜索的 IAM 策略是 “cloudwatch:PutMetricData”。要添加的策略如下所示：

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData"
             ],
             "Effect": "Allow",
             "Resource": "*"
        }
   ]
}
```

> **Tip**
>
> 有关创建 IAM 用户和为添加策略的更多信息 CloudWatch，请参阅 CloudWatch 文档。

2. 在您的 DLAMI 上，运行 Amazon 配置并指定 IAM 用户凭证。

```
$ aws configure
```

3. 您可能需要先对 gpumon 实用工具进行一些修改，然后再运行该工具。您可以在以下代码块中定义的位置中找到 gpumon 实用工具和 README。有关 gpumon.py 脚本的更多信息，请参阅脚本的 Amazon S3 位置。

```
Folder: ~/tools/GPUCloudWatchMonitor
Files:  ~/tools/GPUCloudWatchMonitor/gpumon.py
       ~/tools/GPUCloudWatchMonitor/README
```

选项：

- 如果您的实例不在 us-east-1 中，请在 gpumon.py 中更改区域。
- 更改其他参数，例如 CloudWatchnamespace或报告周期store_reso。

4. 目前，该脚本仅支持 Python 3。激活您的首选框架的 Python 3 环境或激活 DLAMI 一般 Python 3 环境。

```
$ source activate python3
```

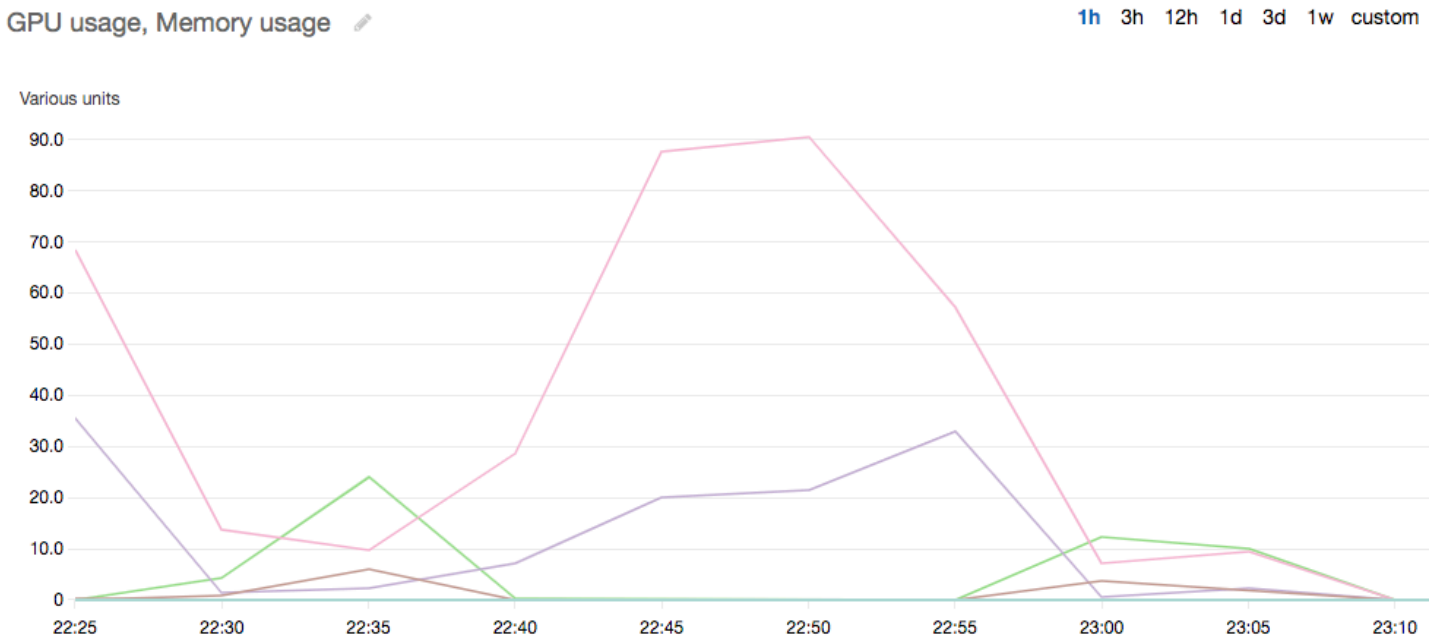
5. 在后台中运行 gpumon 实用工具。

```
(python3)$ python gpumon.py &
```

6. 打开您的浏览器前往 https://console.aws.amazon.com/cloudwatch/，然后选择指标。它将有一个命名空间 “DeepLearningTrain”。

> **Tip**
>
> 您可以修改 gpumon.py 来更改该命名空间。您也可以通过调整 `store_reso` 来修改报告间隔。

以下是一个示例 CloudWatch 图表，报告了 gpumon.py 在监视 p2.8xlarge 实例上的训练作业的情况。



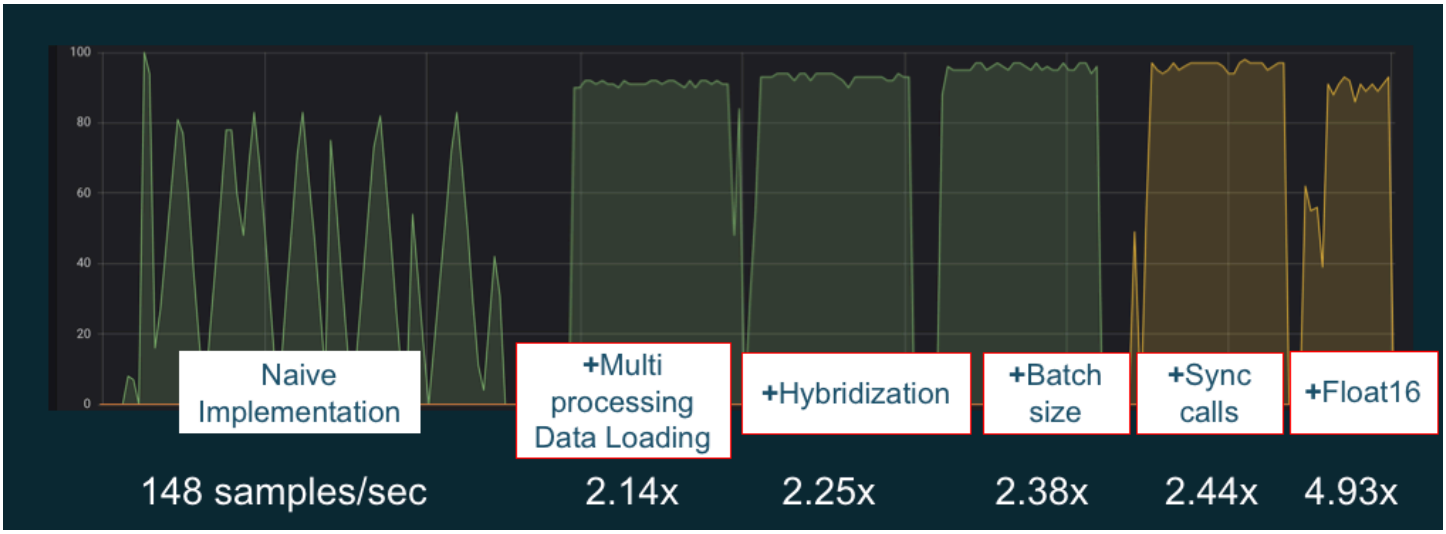
您可能对有关 GPU 监控和优化的以下其他主题感兴趣：

- 监控
  - GPUs 使用监视器 CloudWatch
- 优化
  - 预处理
  - 训练

优化

要充分利用您的数据 GPUs，您可以优化数据管道并调整深度学习网络。如以下图表所述，神经网络的简单或基本实施对 GPU 的使用可能会不一致且不充分。当您优化预处理和数据加载时，您可以从您的 CPU 到 GPU 减少瓶颈。您可以通过使用混合化（该框架支持时）、调整批大小并同步调用来调整神经网络本身。您也可以在大多数框架中使用多精度（float16 或 int8）训练，从而可以显著提高吞吐量。

以下图表显示应用不同优化时的累积性能提升。您的结果将取决于要处理的数据和要优化的网络。



示例 GPU 性能优化。图表来源：使用 MXNet Gluon 的性能技巧

以下指南介绍将使用您的 DLAMI 并帮助您提升 GPU 性能的选项。

主题

- 预处理
- 训练

预处理

通过转换或扩增的数据预处理通常可以是一个绑定 CPU 的流程，而且这可以是您的整体管道中的瓶颈。框架具有用于图像处理的内置运算符，但 DALI（数据扩增库）通过框架的内置选项展示了改进的性能。

- NVIDIA 数据扩增库 (DALI)：DALI 将数据扩增卸载到 GPU。该项未预安装在 DLAMI 上，但您可以通过安装它或在您的 DLAMI 或其他 Amazon Elastic Compute Cloud 实例上加载支持的框架容器来

访问它。有关详细信息，请参阅 NVIDIA 网站上的 DALI 项目页面。有关示例用例和下载代码示例，请参阅 SageMaker 预处理训练性能示例。

- nvJPEG：一个面向 C 编程人员的 GPU 加速型 JPEG 解码器库。它支持解码单个图像或批处理以及深度学习中常见的后续转换操作。nvJPEG 具有内置 DALI，或者您可以从 NVIDIA 网站的 nvjpeg 页面下载并单独使用它。

您可能对有关 GPU 监控和优化的以下其他主题感兴趣：

- 监控
  - GPUs 使用监视器 CloudWatch
- 优化
  - 预处理
  - 训练

训练

利用混合精度训练，您可以使用相同的内存量部署更大的网络，或者减少内存使用量（与您的单精度或双精度网络相比），并且您将看到计算性能增加。您还将受益于更小且更快的数据传输，这在多节点分布式训练中是一个重要因素。要利用混合精度训练，您需要调整数据转换和损失比例。以下是介绍如何针对支持混合精度的框架执行此操作的指南。

- NVIDIA 深度学习 SDK ——NVIDIA 网站上描述了 MXNet PyTorch、和的混合精度实现的文档。TensorFlow

> **Tip**
>
> 请务必针对您选择的框架检查网站，并且搜索“混合精度”或“fp16”，了解最新的优化方法。下面是可能对您有帮助的一些混合精度指南：
>
> - 混合精度训练 TensorFlow （视频）-在 NVIDIA 博客网站上。
> - 使用 float16 进行混合精度训练，其中包含网站 MXNet上的常见问题解答文章。 MXNet
> - NVIDIA Apex：一款用于轻松进行混合精度训练的工具 PyTorch ——NVIDIA 网站上的一篇博客文章。

您可能对有关 GPU 监控和优化的以下其他主题感兴趣：

- 监控
  - GPUs 使用监视器 CloudWatch
- 优化
  - 预处理
  - 训练

带有 DLAMI 的 Amazon 推理芯片

Amazon Inferentia 是一款由其设计的自定义机器学习芯片 Amazon ，可用于高性能的推理预测。要使用该芯片，请设置亚马逊弹性计算云实例，然后使用 Ne Amazon uron 软件开发套件 (SDK) 调用 Inferentia 芯片。为了向客户提供最佳 Inferentia 体验，Neuron 已内置在 Amazon Deep Learning AMIs (DLAMI) 中。

以下主题将向您展示如何开始将 Inferentia 与 DLAMI 结合使用。

内容

- 启动带有神经元的 DLAMI 实例 Amazon
- 将 DLAMI 与神经元一起使用 Amazon

启动带有神经元的 DLAMI 实例 Amazon

最新的 DLAMI 已准备好 Amazon 与 Inferentia 一起使用，并附带 Neuron API 包。 Amazon 要启动 DLAMI 实例，请参阅启动和配置 DLAMI。获得 DLAMI 后，请使用此处的步骤确保 Amazon 您的推理芯片 Amazon 和 Neuron 资源处于活动状态。

内容

- 验证您的实例
- 识别 Amazon 推理设备
- 查看资源使用量
- 使用 Neuron Monitor（Neuron 监视器）
- 升级 Neuron 软件

验证您的实例

在使用您的实例之前，验证该实例是否已针对 Neuron 进行正确的设置和配置。

识别 Amazon 推理设备

要确定实例上的 Inferentia 设备数量，请使用以下命令：

```
neuron-ls
```

如果您的实例已附加了 Inferentia 设备，则输出将如下所示：

```
+--------+--------+--------+-----------+--------------+
| NEURON | NEURON | NEURON | CONNECTED |     PCI      |
| DEVICE | CORES  | MEMORY |  DEVICES  |     BDF      |
+--------+--------+--------+-----------+--------------+
| 0      | 4      | 8 GB   | 1         | 0000:00:1c.0 |
| 1      | 4      | 8 GB   | 2, 0      | 0000:00:1d.0 |
| 2      | 4      | 8 GB   | 3, 1      | 0000:00:1e.0 |
| 3      | 4      | 8 GB   | 2         | 0000:00:1f.0 |
+--------+--------+--------+-----------+--------------+
```

提供的输出取自 Inf1.6xlarge 实例，包括以下各列：

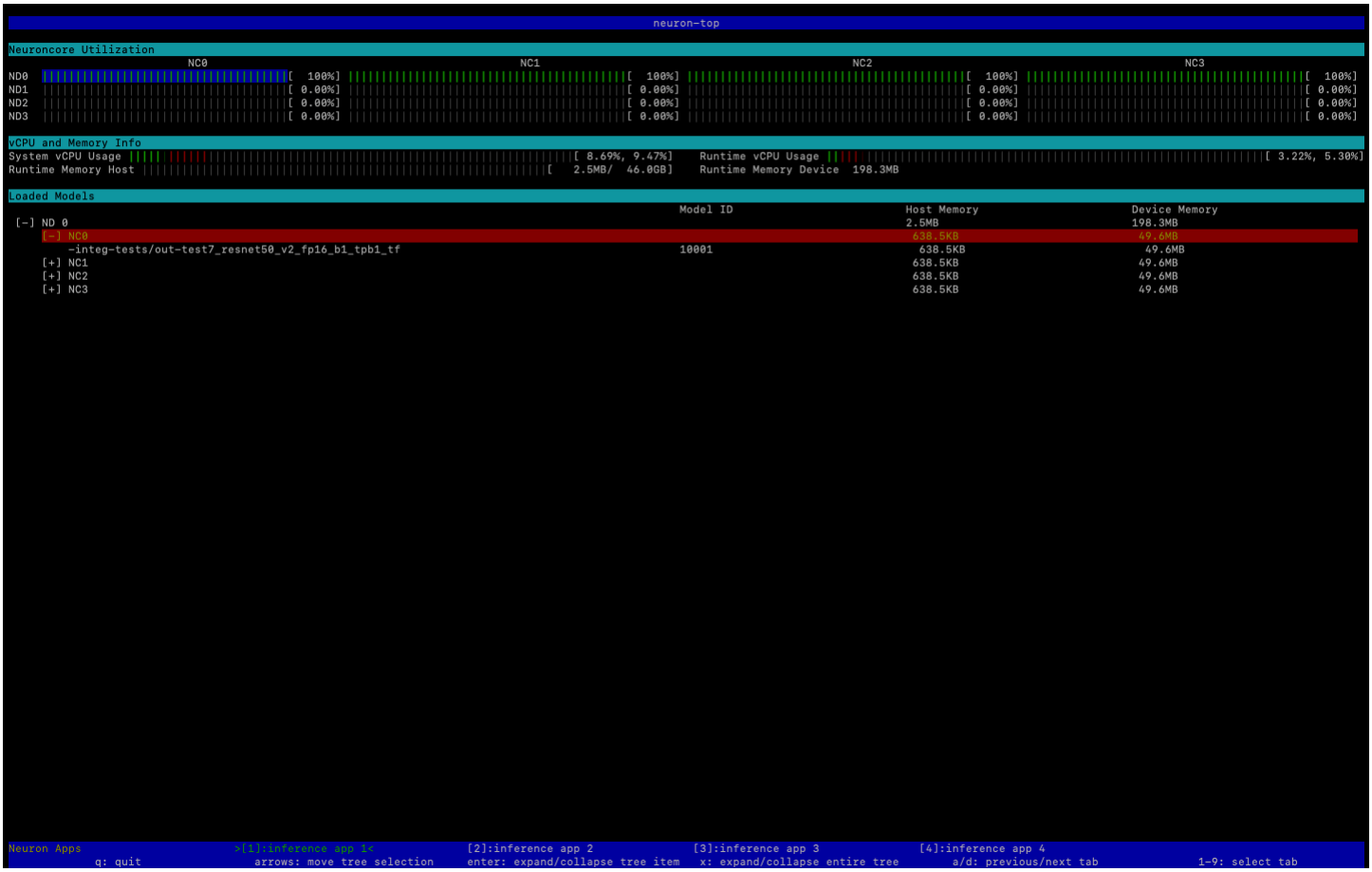
- 神经元设备：分配给的逻辑 ID。 NeuronDevice此 ID 用于将多个运行时配置为使用不同的 NeuronDevices运行时。
- NEURON CORES： NeuronCores 存在于 NEURON CORES 中的 NeuronDevice数量。
- 神经元内存：中的 DRAM 内存量。 NeuronDevice
- 连接的设备：其他 NeuronDevices 连接到 NeuronDevice。
- PCI BDF：的 PCI 总线设备功能 (BDF) ID。 NeuronDevice

查看资源使用量

使用命令查看有关 NeuronCore vCPU 利用率、内存使用率、已加载模型和 Neuron 应用程序的有用信息。`neuron-top`不`neuron-top`带参数启动将显示所有使用的机器学习应用程序的数据 NeuronCores。

```
neuron-top
```

当应用程序使用四时 NeuronCores，输出应类似于下图：



有关用于监控和优化基于 Neuron 的推理应用程序的资源的更多信息，请参阅 Neuron 工具。

使用 Neuron Monitor（Neuron 监视器）

Neuron Monitor 从系统上运行的 Neuron 运行时系统收集指标，并将收集的数据以 JSON 格式流式传输到 stdout。这些指标按指标组进行组织，您可以通过提供配置文件进行配置。有关 Neuron Monitor 的更多信息，请参阅 Neuron Monitor 用户指南。

升级 Neuron 软件

有关如何在 DLAMI 中更新 Neuron SDK 软件的信息，请参阅 Amazon 《神经元设置指南》。

下一个步骤

将 DLAMI 与神经元一起使用 Amazon

将 DLAMI 与神经元一起使用 Amazon

Ne Amazon uron SDK 的典型工作流程是在编译服务器上编译之前训练过的机器学习模型。之后，将构件分发给 Inf1 实例以供执行。 Amazon Deep Learning AMIs (DLAMI) 预装了在使用 Inferentia 的 Inf1 实例中编译和运行推理所需的一切。

以下部分说明如何将 DLAMI 与 Inferentia 结合使用。

内容

使用 TensorFlow-Neuron 和 Neuron 编译器 Amazon

本教程演示如何使用 Ne Amazon uron 编译器编译 Keras ResNet -50 模型并将其以格式导出为已保存的模型。 SavedModel 这种格式是典型的 TensorFlow 模型可互换格式。您还可以学习如何使用示例输入，在 Inf1 实例上运行推理过程。

有关 Neuron SDK 的更多信息，请参阅 Amazon Neuron SDK 文档。

内容

前提条件

使用本教程之前，您应已完成 启动带有神经元的 DLAMI 实例 Amazon 中的设置步骤。您还应该熟悉深度学习知识以及如何使用 DLAMI。

激活 Conda 环境

使用以下命令激活 TensorFlow-Neuron conda 环境：

```
source activate aws_neuron_tensorflow_p36
```

要退出当前 Conda 环境，请运行以下命令：

```
source deactivate
```

Resnet50 编译

创建一个名为 **tensorflow_compile_resnet50.py** 的 Python 脚本，其中包含以下内容。此 Python 脚本编译 Keras ResNet 50 模型并将其导出为已保存的模型。

```
import os
import time
import shutil
import tensorflow as tf
import tensorflow.neuron as tfn
import tensorflow.compat.v1.keras as keras
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

# Create a workspace
WORKSPACE = './ws_resnet50'
os.makedirs(WORKSPACE, exist_ok=True)

# Prepare export directory (old one removed)
model_dir = os.path.join(WORKSPACE, 'resnet50')
compiled_model_dir = os.path.join(WORKSPACE, 'resnet50_neuron')
shutil.rmtree(model_dir, ignore_errors=True)
shutil.rmtree(compiled_model_dir, ignore_errors=True)

# Instantiate Keras ResNet50 model
keras.backend.set_learning_phase(0)
model = ResNet50(weights='imagenet')

# Export SavedModel
tf.saved_model.simple_save(
 session            = keras.backend.get_session(),
 export_dir         = model_dir,
 inputs             = {'input': model.inputs[0]},
```

```
 outputs            = {'output': model.outputs[0]})

# Compile using Neuron
tfn.saved_model.compile(model_dir, compiled_model_dir)

# Prepare SavedModel for uploading to Inf1 instance
shutil.make_archive(compiled_model_dir, 'zip', WORKSPACE, 'resnet50_neuron')
```

使用以下命令编译该模型：

```
python tensorflow_compile_resnet50.py
```

编译过程将需要几分钟时间。完成后，您的输出应与以下内容类似：

```
...
INFO:tensorflow:fusing subgraph neuron_op_d6f098c01c780733 with neuron-cc
INFO:tensorflow:Number of operations in TensorFlow session: 4638
INFO:tensorflow:Number of operations after tf.neuron optimizations: 556
INFO:tensorflow:Number of operations placed on Neuron runtime: 554
INFO:tensorflow:Successfully converted ./ws_resnet50/resnet50 to ./ws_resnet50/
resnet50_neuron
...
```

编译后，保存的模型将进行压缩，放置在 **ws_resnet50/resnet50_neuron.zip** 中。使用以下命令对模型解压缩，并下载用于推理的示例图像：

```
unzip ws_resnet50/resnet50_neuron.zip -d .
curl -O https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg
```

ResNet50 推论

创建一个名为 **tensorflow_infer_resnet50.py** 的 Python 脚本，其中包含以下内容。此脚本使用先前编译的推理模型，对下载的模型运行推理过程。

```
import os
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import resnet50

# Create input from image
img_sgl = image.load_img('kitten_small.jpg', target_size=(224, 224))
img_arr = image.img_to_array(img_sgl)
img_arr2 = np.expand_dims(img_arr, axis=0)
img_arr3 = resnet50.preprocess_input(img_arr2)
# Load model
COMPILED_MODEL_DIR = './ws_resnet50/resnet50_neuron/'
predictor_inferentia = tf.contrib.predictor.from_saved_model(COMPILED_MODEL_DIR)
# Run inference
model_feed_dict={'input': img_arr3}
infa_rslts = predictor_inferentia(model_feed_dict);
# Display results
print(resnet50.decode_predictions(infa_rslts["output"], top=5)[0])
```

使用以下命令对模型运行推理过程：

```
python tensorflow_infer_resnet50.py
```

您的输出应与以下内容类似：

```
...
[('n02123045', 'tabby', 0.6918919), ('n02127052', 'lynx', 0.12770271), ('n02123159',
 'tiger_cat', 0.08277027), ('n02124075', 'Egyptian_cat', 0.06418919), ('n02128757',
 'snow_leopard', 0.009290541)]
```

下一个步骤

使用 Amazon 神经元服务 TensorFlow

使用 Amazon 神经元服务 TensorFlow

本教程展示了在导出保存的模型以用 Amazon 于 Serving 之前，如何构造图形并添加 Neuron 编译步骤 TensorFlow 。 TensorFlow Serving 是一种服务系统，允许您在网络上扩大推理规模。神经元 TensorFlow 服务使用与普通 TensorFlow 服务相同的 API。唯一的区别是，必须为 Amazon Inferentia 编译保存的模型，并且入口点是名为的不同二进制文件。`tensorflow_model_server_neuron`二

进制文件位于 /usr/local/bin/tensorflow_model_server_neuron 中，并已预安装在 DLAMI 中。

有关 Neuron SDK 的更多信息，请参阅 Amazon Neuron SDK 文档。

内容

前提条件

使用本教程之前，您应已完成 启动带有神经元的 DLAMI 实例 Amazon 中的设置步骤。您还应该熟悉深度学习知识以及如何使用 DLAMI。

激活 Conda 环境

使用以下命令激活 TensorFlow-Neuron conda 环境：

```
source activate aws_neuron_tensorflow_p36
```

如果需要退出当前 Conda 环境，请运行：

```
source deactivate
```

编译和导出保存的模型

创建一个名 tensorflow-model-server-compile.py 为的 Python 脚本，其中包含以下内容。该脚本构造一个图形并使用 Neuron 对其进行编译。然后，它将编译的图形导出为保存的模型。

```
import tensorflow as tf
import tensorflow.neuron
import os
```

```
tf.keras.backend.set_learning_phase(0)
model = tf.keras.applications.ResNet50(weights='imagenet')
sess = tf.keras.backend.get_session()
inputs = {'input': model.inputs[0]}
outputs = {'output': model.outputs[0]}

# save the model using tf.saved_model.simple_save
modeldir = "./resnet50/1"
tf.saved_model.simple_save(sess, modeldir, inputs, outputs)

# compile the model for Inferentia
neuron_modeldir = os.path.join(os.path.expanduser('~'), 'resnet50_inf1', '1')
tf.neuron.saved_model.compile(modeldir, neuron_modeldir, batch_size=1)
```

使用以下命令编译该模型：

```
python tensorflow-model-server-compile.py
```

您的输出应与以下内容类似：

```
...
INFO:tensorflow:fusing subgraph neuron_op_d6f098c01c780733 with neuron-cc
INFO:tensorflow:Number of operations in TensorFlow session: 4638
INFO:tensorflow:Number of operations after tf.neuron optimizations: 556
INFO:tensorflow:Number of operations placed on Neuron runtime: 554
INFO:tensorflow:Successfully converted ./resnet50/1 to /home/ubuntu/resnet50_inf1/1
```

处理保存的模型

当模型编译完成后，您可以使用以下命令，通过 tensorflow_model_server_neuron 二进制文件处理保存的模型：

```
tensorflow_model_server_neuron --model_name=resnet50_inf1 \
    --model_base_path=$HOME/resnet50_inf1/ --port=8500 &
```

您的输出应与以下内容类似。编译的模型由服务器暂存在 Inferentia 设备的 DRAM 中，以准备好执行推理过程。

```
...
2019-11-22 01:20:32.075856: I external/org_tensorflow/tensorflow/cc/saved_model/
loader.cc:311] SavedModel load for tags { serve }; Status: success. Took 40764
 microseconds.
2019-11-22 01:20:32.075888: I tensorflow_serving/servables/tensorflow/
saved_model_warmup.cc:105] No warmup data file found at /home/ubuntu/resnet50_inf1/1/
assets.extra/tf_serving_warmup_requests
2019-11-22 01:20:32.075950: I tensorflow_serving/core/loader_harness.cc:87]
 Successfully loaded servable version {name: resnet50_inf1 version: 1}
2019-11-22 01:20:32.077859: I tensorflow_serving/model_servers/
server.cc:353] Running gRPC ModelServer at 0.0.0.0:8500 ...
```

生成发送给模型服务器的推理请求

创建一个名为 tensorflow-model-server-infer.py 的 Python 脚本，其中包含以下内容。该脚本通过 GRPC（这是一个服务框架）运行推理过程。

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
    channel = grpc.insecure_channel('localhost:8500')
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    img_file = tf.keras.utils.get_file(
        "./kitten_small.jpg",
        "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
    img = image.load_img(img_file, target_size=(224, 224))
    img_array = preprocess_input(image.img_to_array(img)[None, ...])
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'resnet50_inf1'
    request.inputs['input'].CopyFrom(
        tf.contrib.util.make_tensor_proto(img_array, shape=img_array.shape))
    result = stub.Predict(request)
    prediction = tf.make_ndarray(result.outputs['output'])
```

```
    print(decode_predictions(prediction))
```

通过使用 GRPC 以及以下命令，在模型上运行推理过程：

```
python tensorflow-model-server-infer.py
```

您的输出应与以下内容类似：

```
[[('n02123045', 'tabby', 0.6918919), ('n02127052', 'lynx', 0.12770271), ('n02123159',
 'tiger_cat', 0.08277027), ('n02124075', 'Egyptian_cat', 0.06418919), ('n02128757',
 'snow_leopard', 0.009290541)]]
```

使用 MXNet-Neuron 和 Neuron 编译器 Amazon

MXNet-Neuron 编译 API 提供了一种编译模型图的方法，您可以在 Amazon Inferentia 设备上运行该模型。

在此示例中，您使用 API 编译 ResNet -50 模型并使用它来运行推理。

有关 Neuron SDK 的更多信息，请参阅 Amazon Neuron SDK 文档。

内容

前提条件

使用本教程之前，您应已完成 启动带有神经元的 DLAMI 实例 Amazon 中的设置步骤。您还应该熟悉深度学习知识以及如何使用 DLAMI。

激活 Conda 环境

使用以下命令激活 MXNet-Neuron conda 环境：

```
source activate aws_neuron_mxnet_p36
```

要退出当前 Conda 环境，请运行：

```
source deactivate
```

Resnet50 编译

创建一个名为 **mxnet_compile_resnet50.py** 的 Python 脚本，其中包含以下内容。此脚本使用 MXNet-Neuron 编译 Python API 来编译 ResNet -50 模型。

```
import mxnet as mx
import numpy as np

print("downloading...")
path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
print("download finished.")

sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)

print("compile for inferentia using neuron... this will take a few minutes...")
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32') }

sym, args, aux = mx.contrib.neuron.compile(sym, args, aux, inputs)

print("save compiled model...")
mx.model.save_checkpoint("compiled_resnet50", 0, sym, args, aux)
```

使用以下命令编译该模型：

```
python mxnet_compile_resnet50.py
```

编译需要几分钟。当编译完成后，以下文件将出现在您的当前目录中：

```
resnet-50-0000.params
resnet-50-symbol.json
compiled_resnet50-0000.params
```

```
compiled_resnet50-symbol.json
```

ResNet50 推论

创建一个名为 **mxnet_infer_resnet50.py** 的 Python 脚本，其中包含以下内容。此脚本会下载一个示例映像，然后使用该映像对已编译的模型运行推理过程。

```
import mxnet as mx
import numpy as np

path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'synset.txt')

fname = mx.test_utils.download('https://raw.githubusercontent.com/awslabs/mxnet-model-
server/master/docs/images/kitten_small.jpg')
img = mx.image.imread(fname)

# convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224)
# resize
img = img.transpose((2, 0, 1))
# Channel first
img = img.expand_dims(axis=0)
# batchify
img = img.astype(dtype='float32')

sym, args, aux = mx.model.load_checkpoint('compiled_resnet50', 0)
softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax
args['data'] = img
# Inferentia context
ctx = mx.neuron()

exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')
with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]

exe.forward(data=img)
prob = exe.outputs[0].asnumpy()
# print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[::-1]
```

```
for i in a[0:5]:
    print('probability=%f, class=%s' %(prob[i], labels[i]))
```

使用以下命令对已编译模型运行推理过程：

```
python mxnet_infer_resnet50.py
```

您的输出应与以下内容类似：

```
probability=0.642454, class=n02123045 tabby, tabby cat
probability=0.189407, class=n02123159 tiger cat
probability=0.100798, class=n02124075 Egyptian cat
probability=0.030649, class=n02127052 lynx, catamount
probability=0.016278, class=n02129604 tiger, Panthera tigris
```

下一个步骤

使用 MXNet神经元模型服务

使用 MXNet神经元模型服务

在本教程中，您将学习使用预训练的 MXNet 模型通过多模型服务器 (MMS) 执行实时图像分类。MMS 是一种灵活的 easy-to-use工具，用于提供使用任何机器学习或深度学习框架训练的深度学习模型。本教程包括使用 Amazon Neuron 的编译步骤和使用 MMS 的实现。 MXNet

有关 Neuron SDK 的更多信息，请参阅 Amazon Neuron SDK 文档。

内容

- 前提条件
- 激活 Conda 环境
- 下载示例代码
- 编译模型
- 运行推理

前提条件

使用本教程之前，您应已完成 启动带有神经元的 DLAMI 实例 Amazon 中的设置步骤。您还应该熟悉深度学习知识以及如何使用 DLAMI。

激活 Conda 环境

使用以下命令激活 MXNet-Neuron conda 环境：

```
source activate aws_neuron_mxnet_p36
```

要退出当前 Conda 环境，请运行：

```
source deactivate
```

下载示例代码

要运行本示例，请使用以下命令下载示例代码：

```
git clone https://github.com/awslabs/multi-model-server
cd multi-model-server/examples/mxnet_vision
```

编译模型

创建一个名为 `multi-model-server-compile.py` 的 Python 脚本，其中包含以下内容。此脚本将 ResNet 50 模型编译为 Inferentia 设备目标。

```
import mxnet as mx
from mxnet.contrib import neuron
import numpy as np

path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
mx.test_utils.download(path+'synset.txt')

nn_name = "resnet-50"

#Load a model
sym, args, auxs = mx.model.load_checkpoint(nn_name, 0)

#Define compilation parameters#  - input shape and dtype
inputs = {'data' : mx.nd.zeros([1,3,224,224], dtype='float32') }

# compile graph to inferentia target
csym, cargs, cauxs = neuron.compile(sym, args, auxs, inputs)
```

```
# save compiled model
mx.model.save_checkpoint(nn_name + "_compiled", 0, csym, cargs, cauxs)
```

要编译模型，请使用以下命令：

```
python multi-model-server-compile.py
```

您的输出应与以下内容类似：

```
...
[21:18:40] src/nnvm/legacy_json_util.cc:209: Loading symbol saved by previous version
 v0.8.0. Attempting to upgrade...
[21:18:40] src/nnvm/legacy_json_util.cc:217: Symbol successfully upgraded!
[21:19:00] src/operator/subgraph/build_subgraph.cc:698: start to execute partition
 graph.
[21:19:00] src/nnvm/legacy_json_util.cc:209: Loading symbol saved by previous version
 v0.8.0. Attempting to upgrade...
[21:19:00] src/nnvm/legacy_json_util.cc:217: Symbol successfully upgraded!
```

创建一个名为 `signature.json` 的文件，其中包含以下内容，以便配置输入名称和形状：

```
{
  "inputs": [
    {
      "data_name": "data",
      "data_shape": [
        1,
        3,
        224,
        224
      ]
    }
  ]
}
```

使用以下命令下载 `synset.txt` 文件。此文件是 ImageNet 预测类的名称列表。

```
curl -O https://s3.amazonaws.com/model-server/model_archive_1.0/examples/
squeezenet_v1.1/synset.txt
```

基于 model_server_template 文件夹中的模板，创建自定义服务类。使用以下命令，将模板复制到您的当前工作目录中：

```
cp -r ../model_service_template/* .
```

编辑 mxnet_model_service.py 模块，将 mx.cpu() 上下文替换为 mx.neuron() 上下文，如下所示。你还需要注释掉不必要的数据副本，model_input因为 MXNet-Neuron 不支持 NDArray 和 Gluon。 APIs

```
...
self.mxnet_ctx = mx.neuron() if gpu_id is None else mx.gpu(gpu_id)
...
#model_input = [item.as_in_context(self.mxnet_ctx) for item in model_input]
```

使用以下命令，通过模型归档程序对模型进行打包：

```
cd ~/multi-model-server/examples
model-archiver --force --model-name resnet-50_compiled --model-path mxnet_vision --
handler mxnet_vision_service:handle
```

运行推理

启动多模型服务器并使用以下命令加载使用 RESTful API 的模型。确保 neuron-rtd 正在使用默认设置运行。

```
cd ~/multi-model-server/
multi-model-server --start --model-store examples > /dev/null # Pipe to log file if you
 want to keep a log of MMS
curl -v -X POST "http://localhost:8081/models?
initial_workers=1&max_workers=4&synchronous=true&url=resnet-50_compiled.mar"
sleep 10 # allow sufficient time to load model
```

通过以下命令，使用示例图像运行推理：

```
curl -O https://raw.githubusercontent.com/awslabs/multi-model-server/master/docs/
images/kitten_small.jpg
curl -X POST http://127.0.0.1:8080/predictions/resnet-50_compiled -T kitten_small.jpg
```

您的输出应与以下内容类似：

```
[
```

```
  {
    "probability": 0.6388034820556641,
    "class": "n02123045 tabby, tabby cat"
  },
  {
    "probability": 0.16900072991847992,
    "class": "n02123159 tiger cat"
  },
  {
    "probability": 0.12221276015043259,
    "class": "n02124075 Egyptian cat"
  },
  {
    "probability": 0.028706775978207588,
    "class": "n02127052 lynx, catamount"
  },
  {
    "probability": 0.01915954425930977,
    "class": "n02129604 tiger, Panthera tigris"
  }
]
```

要在测试结束后进行清理，请通过 RESTful API 发出删除命令并使用以下命令停止模型服务器：

```
curl -X DELETE http://127.0.0.1:8081/models/resnet-50_compiled

multi-model-server --stop
```

您应看到以下输出：

```
{
  "status": "Model \"resnet-50_compiled\" unregistered"
}
Model server stopped.
Found 1 models and 1 NCGs.
Unloading 10001 (MODEL_STATUS_STARTED) :: success
Destroying NCG 1 :: success
```

使用 PyTorch-Neuron 和 Neuron 编译器 Amazon

PyTorch-Neuron 编译 API 提供了一种编译模型图的方法，您可以在 Amazon Inferentia 设备上运行该模型。

经过训练的模型必须先编译为 Inferentia 目标，才能部署在 Inf1 实例上。以下教程编译 torchvision ResNet 50 模型并将其导出为已保存的模块。 TorchScript 此模型随后将用于运行推理。

为方便起见，本教程使用 Inf1 实例进行编译和推理。在实际操作中，您也可以使用其他实例类型来编译模型，例如 c5 实例系列。然后，您必须将已编译的模型部署到 Inf1 推理服务器中。有关更多信息，请参阅 Ne Amazon uron PyTorch SDK 文档。

内容

前提条件

使用本教程之前，您应已完成 启动带有神经元的 DLAMI 实例 Amazon 中的设置步骤。您还应该熟悉深度学习知识以及如何使用 DLAMI。

激活 Conda 环境

使用以下命令激活 PyTorch-Neuron conda 环境：

```
source activate aws_neuron_pytorch_p36
```

要退出当前 Conda 环境，请运行：

```

source deactivate
```

Resnet50 编译

创建一个名为 **`pytorch_trace_resnet50.py`** 的 Python 脚本，其中包含以下内容。此脚本使用 PyTorch-Neuron 编译 Python API 来编译 ResNet -50 模型。

> **Note**
>
> 在编译 torchvision 模型时，您需要注意：torchvision 与 torch 软件包的版本之间存在依赖关系。这些依赖关系规则可以通过 pip 进行管理。Torchvision==0.6.1 与 torch==1.5.1 版本匹配，而 torchvision==0.8.2 与 torch==1.7.1 版本匹配。

```
import torch
import numpy as np
import os
import torch_neuron
from torchvision import models

image = torch.zeros([1, 3, 224, 224], dtype=torch.float32)

## Load a pretrained ResNet50 model
model = models.resnet50(pretrained=True)

## Tell the model we are using it for evaluation (not training)
model.eval()
model_neuron = torch.neuron.trace(model, example_inputs=[image])

## Export to saved model
model_neuron.save("resnet50_neuron.pt")
```

运行编译脚本。

```
python pytorch_trace_resnet50.py
```

编译需要几分钟。编译完成后，已编译的模型将以 `resnet50_neuron.pt` 的形式保存在本地目录中。

ResNet50 推论

创建一个名为 **`pytorch_infer_resnet50.py`** 的 Python 脚本，其中包含以下内容。此脚本会下载一个示例映像，然后使用该映像对已编译的模型运行推理过程。

```
import os
import time
import torch
import torch_neuron
import json
import numpy as np

from urllib import request

from torchvision import models, transforms, datasets
```

```
## Create an image directory containing a small kitten
os.makedirs("./torch_neuron_test/images", exist_ok=True)
request.urlretrieve("https://raw.githubusercontent.com/awslabs/mxnet-model-server/
master/docs/images/kitten_small.jpg",
                    "./torch_neuron_test/images/kitten_small.jpg")


## Fetch labels to output the top classifications
request.urlretrieve("https://s3.amazonaws.com/deep-learning-models/image-models/
imagenet_class_index.json","imagenet_class_index.json")
idx2label = []

with open("imagenet_class_index.json", "r") as read_file:
    class_idx = json.load(read_file)
    idx2label = [class_idx[str(k)][1] for k in range(len(class_idx))]

## Import a sample image and normalize it into a tensor
normalize = transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225])

eval_dataset = datasets.ImageFolder(
    os.path.dirname("./torch_neuron_test/"),
    transforms.Compose([
    transforms.Resize([224, 224]),
    transforms.ToTensor(),
    normalize,
    ])
)

image, _ = eval_dataset[0]
image = torch.tensor(image.numpy()[np.newaxis, ...])

## Load model
model_neuron = torch.jit.load( 'resnet50_neuron.pt' )

## Predict
results = model_neuron( image )

# Get the top 5 results
top5_idx = results[0].sort()[1][-5:]

# Lookup and print the top 5 labels
top5_labels = [idx2label[idx] for idx in top5_idx]
```

```
print("Top 5 labels:\n {}".format(top5_labels) )
```

使用以下命令对已编译模型运行推理过程：

```
python pytorch_infer_resnet50.py
```

您的输出应与以下内容类似：

```
Top 5 labels:
 ['tiger', 'lynx', 'tiger_cat', 'Egyptian_cat', 'tabby']
```

ARM64 DLAMI

Amazon ARM64 GPU 旨在 DLAMIs 为深度学习工作负载提供高性能和成本效益。具体而言，g5G 实例类型采用基于 ARM64 的 G Amazon raviton2 处理器，该处理器是从头开始构建的， Amazon 并针对客户在云中运行工作负载的方式进行了优化。 Amazon ARM64 GPU DLAMIs 预先配置了 Docker、NVIDIA Docker、NVIDIA Driver、CUDA、cuDNN、NCCL 以及流行的机器学习框架，例如和。 TensorFlow PyTorch

借助 G5g 实例类型，您可以利用 Graviton2 的价格和性能优势来部署基于 GPU 加速的深度学习模型，与基于 x86 的带有 GPU 加速的实例相比，成本大幅降低。

选择一个 ARM64 DLAMI

使用您选择的 D ARM64 LAMI 启动 g5G 实例。

有关启动 DLAMI 的 step-by-step说明，请参阅启动和配置 DLAMI。

有关最新版本的列表 ARM64 DLAMIs，请参阅 DLAMI 发行说明。

开始使用

以下主题向您展示如何开始使用 ARM64 DLAMI。

内容

- 使用 ARM64 GPU PyTorch DLAMI

使用 ARM64 GPU PyTorch DLAMI

已准备好在基 Amazon Deep Learning AMIs 于 Arm64 处理器的情况下使用 GPUs，并针对以下方面进行了优化。 PyTorch ARM64 GPU PyTorch DLAMI 包括一个预先配置了、和的 Python 环境 TorchVision，用于深度学习训练TorchServe和推理PyTorch用例。

内容

验证 PyTorch Python 环境

使用以下命令来连接您的 G5g 实例并激活基础 Conda 环境：

```
source activate base
```

您的命令提示符应表明您正在基本 Conda 环境中工作，该环境包含 PyTorch TorchVision、和其他库。

```
(base) $
```

验证 PyTorch 环境的默认刀具路径：

```
(base) $ which python
(base) $ which pip
(base) $ which conda
(base) $ which mamba
>>> import torch, torchvision
>>> torch.__version__
>>> torchvision.__version__
>>> v = torch.autograd.Variable(torch.randn(10, 3, 224, 224))
>>> v = torch.autograd.Variable(torch.randn(10, 3, 224, 224)).cuda()
>>> assert isinstance(v, torch.Tensor)
```

使用运行训练示例 PyTorch

运行示例 MNIST 训练作业：

```
git clone https://github.com/pytorch/examples.git
```

```
cd examples/mnist
python main.py
```

您的输出应类似于以下内容：

```
...
Train Epoch: 14 [56320/60000 (94%)]    Loss: 0.021424
Train Epoch: 14 [56960/60000 (95%)]    Loss: 0.023695
Train Epoch: 14 [57600/60000 (96%)]    Loss: 0.001973
Train Epoch: 14 [58240/60000 (97%)]    Loss: 0.007121
Train Epoch: 14 [58880/60000 (98%)]    Loss: 0.003717
Train Epoch: 14 [59520/60000 (99%)]    Loss: 0.001729
Test set: Average loss: 0.0275, Accuracy: 9916/10000 (99%)
```

使用运行推理示例 PyTorch

使用以下命令下载预训练的 densenet161 模型并使用以下命令运行推理： TorchServe

```
# Set up TorchServe
cd $HOME
git clone https://github.com/pytorch/serve.git
mkdir -p serve/model_store
cd serve

# Download a pre-trained densenet161 model
wget https://download.pytorch.org/models/densenet161-8d451a50.pth >/dev/null

# Save the model using torch-model-archiver
torch-model-archiver --model-name densenet161 \
    --version 1.0 \
    --model-file examples/image_classifier/densenet_161/model.py \
    --serialized-file densenet161-8d451a50.pth \
    --handler image_classifier \
    --extra-files examples/image_classifier/index_to_name.json  \
    --export-path model_store

# Start the model server
torchserve --start --no-config-snapshots \
    --model-store model_store \
    --models densenet161=densenet161.mar &> torchserve.log

# Wait for the model server to start
sleep 30
```

```
# Run a prediction request
curl http://127.0.0.1:8080/predictions/densenet161 -T examples/image_classifier/
kitten.jpg
```

您的输出应类似于以下内容：

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023522898778133094
}
```

使用以下命令来注销 densenet161 模型并停止服务器：

```
curl -X DELETE http://localhost:8081/models/densenet161/1.0
torchserve --stop
```

您的输出应类似于以下内容：

```
{
  "status": "Model \"densenet161\" unregistered"
}
TorchServe has stopped.
```

推理

此部分提供了有关如何使用 DLAMI 的框架和工具来运行推理的教程。

推理工具

- TensorFlow 服务

模型处理

以下是已在带 Conda 的深度学习 AMI 上安装的模型处理选项。单击其中一个选项可了解如何使用该选项。

主题

- TensorFlow 服务
- TorchServe

TensorFlow 服务

TensorFlow S@@ er ving 是一款适用于机器学习模型的灵活、高性能的服务系统。

预装了`tensorflow-serving-api`单框架 DLAMI。要使用 tensorflow 服务，请先激活环境。TensorFlow

```
$ source /opt/tensorflow/bin/activate
```

然后，使用您的首选文本编辑器创建具有以下内容的脚本。将它命名为 `test_train_mnist.py`。此脚本引自TensorFlow 教程，该教程将训练和评估对图像进行分类的神经网络机器学习模型。

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

现在，运行将服务器位置和端口以及哈士奇照片的文件名作为参数传递的脚本。

```
$ /opt/tensorflow/bin/python3 test_train_mnist.py
```

请耐心等待，因为此脚本可能需要一段时间才能提供输出。培训完成后，您应该会看到以下内容：

```
I0000 00:00:1739482012.389276    4284 device_compiler.h:188] Compiled cluster using
 XLA!  This line is logged at most once for the lifetime of the process.
1875/1875 [==============================] - 24s 2ms/step - loss: 0.2973 - accuracy:
 0.9134
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1422 - accuracy:
 0.9582
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1076 - accuracy:
 0.9687
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0872 - accuracy:
 0.9731
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0731 - accuracy:
 0.9771
313/313 [==============================] - 0s 1ms/step - loss: 0.0749 - accuracy:
 0.9780
```

更多功能和示例

如果您有兴趣了解有关 TensorFlow 服务的更多信息，请TensorFlow 访问该网站。

TorchServe

TorchServe 是一款灵活的工具，用于提供已从中导出的深度学习模型 PyTorch。 TorchServe 预装了带有 Conda 的深度学习 AMI。

有关使用的更多信息 TorchServe，请参阅 PyTorch文档模型服务器。

主题

在上提供图像分类模型 TorchServe

本教程介绍如何使用提供图像分类模型 TorchServe。它使用提供的 DenseNet -161 模型。 PyTorch 服务器运行后，它会监听预测请求。在这种情况下，如果您上传图像（一张小猫的图像），服务器会返回在其上训练该模型的类中匹配的前 5 个类的预测。

在上提供图像分类模型示例 TorchServe

1. 使用 Conda v34 或更高版本的深度学习 AMI 连接到亚马逊弹性计算云 (Amazon EC2) 实例。
2. 激活 `pytorch_p310` 环境。

```
source activate pytorch_p310
```

3. 克隆 TorchServe 存储库，然后创建一个目录来存储您的模型。

```
git clone https://github.com/pytorch/serve.git
mkdir model_store
```

4. 使用模型存档程序来存档模型。该extra-files参数使用TorchServe存储库中的文件，因此如有必要，请更新路径。 有关模型存档器的更多信息，请参阅 Torch 模型存档器。 TorchServe

```
wget https://download.pytorch.org/models/densenet161-8d451a50.pth
torch-model-archiver --model-name densenet161 --version 1.0 --model-file ./
serve/examples/image_classifier/densenet_161/model.py --serialized-file
 densenet161-8d451a50.pth --export-path model_store --extra-files ./serve/examples/
image_classifier/index_to_name.json --handler image_classifier
```

5. 运行 TorchServe 以启动终端节点。添加 > /dev/null 会使日志输出静音。

```
torchserve --start --ncs --model-store model_store --models densenet161.mar > /dev/
null
```

6. 下载小猫的图像并将其发送到 TorchServe预测端点：

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
```

该预测终端节点将返回一个 JSON 格式的预测（类似于下面的前 5 个预测），其中，图像具有 47% 的可能性为埃及猫，然后有 46% 的可能性为虎斑猫。

```
{
 "tiger_cat": 0.46933576464653015,
 "tabby": 0.463387668132782,
 "Egyptian_cat": 0.0645613968372345,
 "lynx": 0.0012828196631744504,
 "plastic_bag": 0.00023323058849200606
}
```

7. 当您完成测试时，停止服务器：

```
torchserve --stop
```

其他示例

TorchServe 提供了各种各样的示例，您可以在 DLAMI 实例上运行这些示例。您可以在 TorchServe项目存储库示例页面上查看它们。

更多信息

有关更多 TorchServe 文档，包括如何 TorchServe使用 Docker 进行设置和最新 TorchServe 功能，请参阅上的 TorchServe GitHub项目页面。

升级 DLAMI

在此处，您将找到有关升级 DLAMI 的信息以及有关在 DLAMI 上更新软件的提示。

一旦有修补程序和更新推出便立即应用，从而始终保持操作系统和其他已安装软件为最新。

如果您使用的是 Amazon Linux 或 Ubuntu，则当您登录到您的 DLAMI 时，便会收到更新通知（如果有）并且会看到更新说明。有关 Amazon Linux 维护的更多信息，请参阅更新实例软件。对于 Ubuntu 实例，请参阅官方 Ubuntu 文档。

在 Windows 上，定期检查 Windows Update 有无软件和安全更新。如果您愿意，可以自动应用更新。

> **Important**
>
> 有关 Meltdown 和 Spectre 漏洞以及如何修补操作系统以解决这些漏洞的信息，请参阅安全公告-2018-013。 Amazon

主题

升级到新版 DLAMI

DLAMI 的系统映像定期更新，以利用新的深度学习框架版本、CUDA、其他软件更新和性能优化。如果您已使用 DLAMI 一段时间并想要利用更新，则需要启动新实例。您还必须手动传输任何数据集、检查点或其他宝贵的数据。或者，您可以使用 Amazon EBS 来保留数据，并将其附加到新的 DLAMI。通过这种方法，您可以经常升级，同时最大限度地减少转换数据所需的时间。

> **Note**
>
> 在连接和移动 Amazon EBS 卷时 DLAMIs，必须将 DLAMIs 和新卷都放在同一个可用区中。

1. 使用亚马逊 EC2console 创建新的亚马逊 EBS 卷。有关详细说明，请参阅创建 Amazon EBS 卷。

2. 将新创建的 Amazon EBS 卷附加到现有 DLAMI。有关详细说明，请参阅附加 Amazon EBS 卷。
3. 传输您的数据，如数据集、检查点和配置文件。
4. 启动 DLAMI。有关详细指导，请参阅设置 DLAMI 实例。
5. 从旧 DLAMI 中分离 Amazon EBS 卷。有关详细说明，请参阅分离 Amazon EBS 卷。
6. 将该 Amazon EBS 卷附加到新的 DLAMI。请按照步骤 2 中的相关说明附加卷。
7. 在确认数据可用于新的 DLAMI 上时，停止并终止旧的 DLAMI。有关更详细的清理说明，请参阅清理 DLAMI 实例。

有关软件更新的提示

有时，您可能想要在 DLAMI 上手动更新软件。通常建议您使用 `pip` 来更新 Python 软件包。您还应在带 Conda 的深度学习 AMI 上的 Conda 环境内使用 `pip` 以更新软件包。有关升级和安装说明，请参阅特定框架或软件的网站。

> **Note**
>
> 我们不能保证软件包更新一定成功。尝试在依赖项不兼容的环境中升级软件包可能会导致安装失败。在这种情况下，您应该联系库维护人员，看看是否可以更新软件包依赖项。或者，您可以尝试以允许更新的方式来修改环境。但是，这种修改可能意味着删除或更新现有软件包，这意味着我们无法再保证此环境的稳定性。

它预 Amazon Deep Learning AMIs 装了许多 Conda 环境和许多软件包。由于预装软件包数量众多，要找到一组保证兼容的软件包非常困难。您可能会看到“环境不一致，请仔细检查软件包计划”的警告。DLAMI 确保 DLAMI 提供的所有环境都是正确的，但不能保证用户安装的任何软件包都能正常运行。

有新的更新时收到通知

> **Note**
>
> Amazon 深度学习 AMIs 每周都会发布安全补丁。将针对这些增量安全补丁发送发布通知，尽管它们可能未包含在官方发布说明中。

每当有新的 DLAMI 发布时，您都可以收到通知。通过 Amazon SNS 使用以下主题发布通知。

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

每当有新的 DLAMI 发布时，都将在此处发布消息。AMI 的版本、元数据和区域 AMI ID 将包含在消息中。

可以使用几种不同的方法接收这些消息。我们建议您使用以下方法。

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

安全性 Amazon Deep Learning AMIs

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

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

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

此文档将帮助您了解如何在使用 DLAMI 时应用责任共担模式。以下主题说明如何配置 DLAMI 以实现您的安全性和合规性目标。您还将学习如何使用其他 Amazon Web Services 服务 方法来监控和保护您的DLAMI资源。

有关更多信息，请参阅《亚马逊 EC2 用户指南》 EC2中的 “亚马逊安全”。

主题

中的数据保护 Amazon Deep Learning AMIs

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

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

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

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API Amazon CLI或与 DLAMI Amazon Web Services 服务 或其他人合作时。 Amazon SDKs在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

的身份和访问管理 Amazon Deep Learning AMIs

Amazon Identity and Access Management (IAM) Amazon Web Services 服务 可帮助管理员安全地控制对 Amazon 资源的访问权限。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 DLAMI 资源的人员。您可以使用 IAM Amazon Web Services 服务 ，无需支付额外费用。

有关身份和访问管理的更多信息，请参阅 Amazon 的身份和访问管理 EC2。

主题

- 使用身份进行身份验证
- 使用策略管理访问
- IAM 与 Amazon EMR 结合使用

使用身份进行身份验证

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

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

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， Amazon 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 中的Amazon 多重身份验证。

Amazon Web Services 账户 root 用户

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

IAM 用户和群组

I AM 用户是您 Amazon Web Services 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 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 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 Amazon Web Services 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 Amazon Web Services Management Console，您可以从用

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

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 Amazon Web Services 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解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 中的跨账户资源访问。
- 跨服务访问 — 有些 Amazon Web Services 服务 使用其他 Amazon Web Services 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 Amazon Web Services 服务 向下游服务发出请求的请求。 Amazon Web Services 服务只有当服务收到需要与其他 Amazon Web Services 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 Amazon Web Services 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 Amazon Web Services 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 Amazon Web Services 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 Amazon CLI 或 Amazon API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 Amazon 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含角色并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

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

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

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 Amazon Web Services Management Console Amazon CLI、或 Amazon API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 Amazon Web Services 账户。托管策略包括 Amazon 托管策略和客户托管策略。要了解如何在托管策略和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的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 Amazon Web Services 服务。

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

访问控制列表 (ACLs)

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

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

其他策略类型

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

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 Amazon Organizations。 Amazon Organizations 是一项用于对您的企业拥有的多 Amazon Web Services 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 Amazon Web Services 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《Amazon Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 Amazon Web Services 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 Amazon Web Services 服务 该支持的列表 RCPs，请参阅《Amazon Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

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

IAM 与 Amazon EMR 结合使用

您可以将 IAM 与 Amazon EMR 结合使用来定义用户、 Amazon 资源、群组、角色和策略。您还可以控制 Amazon Web Services 服务 这些用户和角色可以访问哪些用户。

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

合规性验证 Amazon Deep Learning AMIs

Amazon Deep Learning AMIs 作为多个合规计划的一部分，第三方审计师对安全性和 Amazon 合规性进行评估。有关支持的合规计划的信息，请参阅 Amazon 合规性验证 EC2。

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

您可以使用下载第三方审计报告 Amazon Artifact。有关更多信息，请参阅 Downloading reports in Amazon Artifact。

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

- 安全性与合规性快速入门指南安全性与合规性快速入门指南 - 这些部署指南讨论了架构注意事项，并提供了在 Amazon上部署基于安全性和合规性的基准环境的步骤。
- 合规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- 使用Amazon Config 开发人员指南中的 Amazon Config 规则评估资源 — 该 Amazon Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- Amazon Security Hub— 这 Amazon Web Services 服务 提供了您内部安全状态的全面视图 Amazon。Security Hub 使用安全控制来评估您的 Amazon 资源，并检查您是否符合安全行业标准和最佳实践。

韧性在 Amazon Deep Learning AMIs

Amazon 全球基础设施是围绕 Amazon Web Services 区域 可用区构建的。 Amazon Web Services 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础结构相比，可用区具有更高的可用性、容错性和可扩展性。

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

有关有助于满足您的数据弹性和备份需求的 Amazon EC2 功能的信息，请参阅《亚马逊 EC2 用户指南》 EC2中的 Amaz on 弹性。

中的基础设施安全 Amazon Deep Learning AMIs

的基础设施安全由 Amazon 提供支持 EC2。 Amazon Deep Learning AMIs 有关更多信息，请参阅《亚马逊 EC2 用户指南》 EC2中的 Amazon 基础设施安全。

监控 Amazon Deep Learning AMIs 实例

监控是维护您的 Amazon Deep Learning AMIs 实例和其他 Amazon 解决方案的可靠性、可用性和性能的重要组成部分。您的 DLAMI 实例附带了多个 GPU 监控工具，包括向亚马逊报告 GPU 使用情况统计数据的实用工具。 CloudWatch有关更多信息GPU 监控和优化，请参阅亚马逊 EC2 用户指南中的监控亚马逊 EC2 资源。

选择退出 DLAMI 实例的使用情况跟踪

以下 Amazon Deep Learning AMIs 操作系统发行版包括 Amazon 允许收集实例类型、实例 ID、DLAMI 类型和操作系统信息的代码。

> **Note**
>
> Amazon 不会收集或保留有关 DLAMI 的任何其他信息，例如您在 DLAMI 中使用的命令。

- Amazon Linux 2
- Amazon Linux 2023
- Ubuntu 20.04
- Ubuntu 22.04

选择退出使用情况跟踪

如果您选择，则可以选择退出新 DLAMI 实例的使用情况跟踪。要选择退出，您必须在启动期间向 Amazon EC2 实例添加标签。标签应使用密钥 `OPT_OUT_TRACKING`，并将关联值设置为 `true`。有关更多信息，请参阅《亚马逊 EC2 用户指南》中的为您的亚马逊 EC2 资源添加标签。

DLAMI 支持策略

您可以在此处找到 Amazon Deep Learning AMIs (DLAMI) 的支持政策的详细信息。

有关当前支持的DLAMI框架和操作系统的列表 Amazon ，请参阅DLAMI支持政策页面。以下术语适用于 Support 政策页面和本页中 DLAMIs 提及的所有术语：

- 当前版本以 x.y.z 格式指定框架版本。在这种格式中，x 表示主要版本，y 表示次要版本，z 表示补丁版本。例如，对于 TensorFlow 2.10.1，主版本为 2，次要版本为 10，补丁版本为 1。
- 补丁结束指定 Amazon 支持特定框架或操作系统版本的时长。

有关具体内容的详细信息 DLAMIs，请参阅的发行说明 DLAMIs。

DLAMI Support FAQs

- 哪些框架版本会获得安全补丁？
- 哪个操作系统有安全补丁？
- Amazon 发布新框架版本时会发布哪些镜像？
- 哪些图像获得了新的 SageMaker AI/Amazon 功能？
- “支持的框架”表中是如何定义当前版本的？
- 如果我运行的版本不在 “支持” 表中，该怎么办？
- 是否 DLAMIs 支持框架版本的先前补丁版本？
- 如何找到支持的框架版本的最新补丁映像？
- 多长时间发布一次新映像？
- 运行工作负载时，能在我的实例上以替代方式安装补丁吗？
- 如果有新的补丁或更新的框架版本可用，会发生什么呢？
- 是否可在不更改框架版本的情况下更新依赖项？
- 对我的框架版本的主动支持何时结束？
- 对于框架版本不再主动维护的映像，会为其安装补丁吗？
- 如何使用旧框架版本？
- 如何保持框架及其版本 up-to-date的支持变更？
- 是否需要商业许可证才能使用 Anaconda 存储库？

哪些框架版本会获得安全补丁？

如果框架版本位于 Suppor t P olicy 表中的 “Amazon Deep Learning AMIs 支持的框架版本” 下，则会获得安全补丁。

哪个操作系统有安全补丁？

如果操作系统列在 Support P olicy 表的 “支持的操作系统版本” 下，则该Amazon Deep Learning AMIs 操作系统将获得安全补丁。

Amazon 发布新框架版本时会发布哪些镜像？

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

哪些图像获得了新的 SageMaker AI/Amazon 功能？

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

“支持的框架”表中是如何定义当前版本的？

Support P Amazon Deep Learning AMIs olicy 表中的当前版本是指在上提供 Amazon 的最新框架版本 GitHub。每个最新版本都包括对 DLAMI 中驱动程序、库和相关软件包的更新。有关映像维护的信息，请参阅 对我的框架版本的主动支持何时结束？

如果我运行的版本不在 “支持” 表中，该怎么办？

如果您运行的版本不在 Support Amazon Deep Learning AMIs P olicy 表中，则可能没有最新的驱动程序、库和相关包。要获得更多 up-to-date版本，我们建议您使用所选的最新 DLAMI 升级到支持的框架或操作系统。有关可用列表 DLAMIs，请参阅 DLAMI 发行说明。

是否 DLAMIs 支持框架版本的先前补丁版本？

不是。 如Support Policy 表所述，我们支持每个框架最新主要版本的最新补丁版本，自其首次 GitHub 发布起365天后Amazon Deep Learning AMIs 发布。有关更多信息，请参阅 如果我运行的版本不在 “支持” 表中，该怎么办？。

如何找到支持的框架版本的最新补丁映像？

要使用最新框架版本的 DLAMI，您可以使用 Amazon CLI 或 SSM 参数来检索 DLAMI ID，然后使用它通过控制台启动 DLAMI。EC2 有关检索 Amazon Deep Learning AMIs ID 的 Amazon CLI 或 SSM 参数命令示例，请参阅 DLAMI 发行说明页面单框架 DLAMI 发行说明。您选择的框架版本必须列在 Support P olicy 表的 “Amazon Deep Learning AMIs 支持的框架版本” 下。

多长时间发布一次新映像？

提供更新的补丁版本是我们的首要任务。我们通常会尽早创建安装了补丁的映像。我们会监控新修补的框架版本（例如 TensorFlow 2.9 到 TensorFlow 2.9.1）和新的次要发行版本（例如 TensorFlow 2.9 到 TensorFlow 2.10），并尽早提供它们。当现有版本与新版本 TensorFlow 的 CUDA 一起发布时，我们会为该版本发布支持新 CUDA 版本 TensorFlow 的新 DLAMI。

运行工作负载时，能在我的实例上以替代方式安装补丁吗？

不能。DLAMI 的补丁更新不是“替代”更新。

您必须开启新 EC2 实例，迁移工作负载和脚本，然后关闭之前的实例。

如果有新的补丁或更新的框架版本可用，会发生什么呢？

要收到有关 DLAMI 变更的通知，请订阅相关 DLAMI 的通知，请参阅接收有关新更新的通知。

是否可在不更改框架版本的情况下更新依赖项？

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

对我的框架版本的主动支持何时结束？

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

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

> **Note**
>
> 由于版本补丁升级和安全补丁的频率很高，我们建议您经常查看 DLAMI 发布说明页面，并在发生更改时进行升级。

框架版本（补丁）升级

如果你有一个基于 2.7.0 的 DLAMI 工作负载 TensorFlow并在版本为 2.7.1 之后发布，那么 TensorFlow Amazon 就要发布一个带有 2.7.1 GitHub 版本的新 DLAMI。 TensorFlow 2.7.1 版本的新镜像发布后，将不再主动维护之前的 TensorFlow 2.7.0 镜像。2.7.0 版本的 DLAMI TensorFlow 没有收到更多补丁。然后，2.7版的DLAMI发行说明页面将更新 TensorFlow 为最新信息。没有为每个次要补丁提供单独的发布说明页面。

由于补丁升级而 DLAMIs 创建的新用户将使用新的 AMI ID 进行指定。

Amazon 安全补丁

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

由于补丁升级而 DLAMIs 创建的新用户将使用新的 AMI ID 进行指定。

补丁结束日期（已过期）

DLAMIs 在 GitHub 发布日期 365 天后，他们就到了补丁的终止日期。

对于多框架 DLAMIs，当其中一个框架版本更新时，需要更新版本的新 DLAMI。不再主动维护使用旧框架版本的 DLAMI。

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

依赖关系 end-of-support

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

对于框架版本不再主动维护的映像，会为其安装补丁吗？

不会。不再主动维护的图像就不会有新版本。

如何使用旧框架版本？

要使用带有旧框架版本的 DLAMI，请检索 DLAMI ID，然后使用它通过控制台启动 DLAMI。EC2 有关检索 AM Amazon I ID 的 CLI 命令，请参阅单框架 DLAMI 发行说明中的发行说明页面。

如何保持框架及其版本 up-to-date的支持变更？

up-to-date使用 DLAMI 发行说明中的 Framework Su pport Policy 表，继续使用 D LAMI Amazon Deep Learning AMIs 框架和版本。

是否需要商业许可证才能使用 Anaconda 存储库？

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

重要的 NVIDIA 驱动程序更改为 DLAMIs

2023年11月15日，对与使用的NIVIDA驱动程序相关的 Amazon Deep Learning AMIs （DLAMI） Amazon 进行了重要更改。 DLAMIs 有关更改内容以及更改是否会影响您的使用情况的信息 DLAMIs，请参阅DLAMI NVIDIA 驱动程序变更 FAQs。

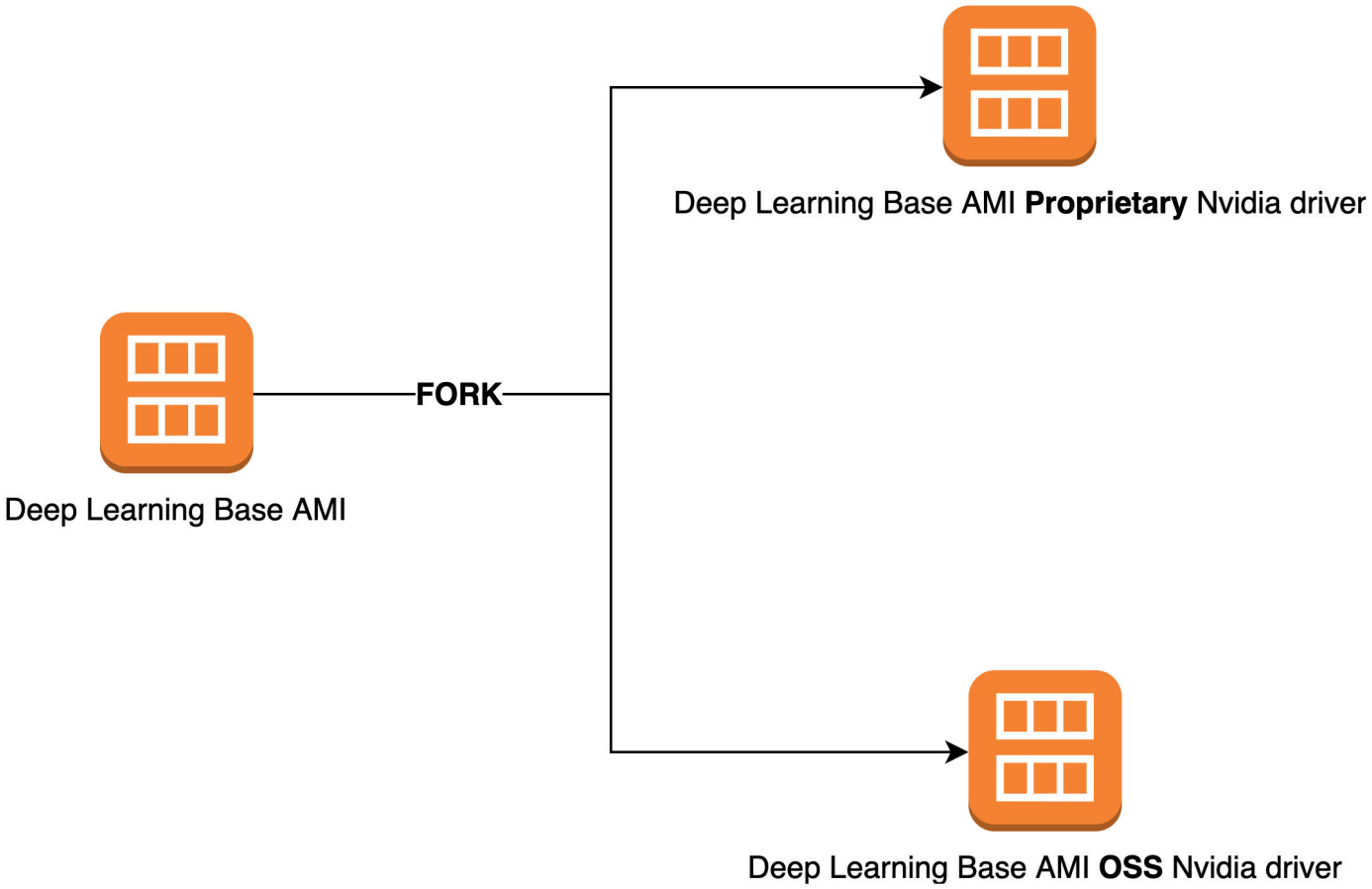
DLAMI NVIDIA 驱动程序变更 FAQs

更改了哪些内容？

我们分 DLAMIs 成两个独立的小组：

- DLAMIs 使用 NVIDIA 专有驱动程序（支持 P3、p3dn、G3）
- DLAMIs 使用 NVIDIA OSS 驱动程序（支持 g4dn、G5、P4、P5）

因此，我们使用新名称和新 AMI DLAMIs 为这两个类别分别创建了新类别 IDs。 DLAMIs 它们不可互换。也就是说， DLAMIs 来自一个组的实例不支持另一个组支持的实例。例如，支持 P5 的 DLAMI 不支持 G3，而支持 G3 的 DLAMI 不支持 P5。



为什么需要进行此更改？

以前， DLAMIs NVIDIA GPUs 包含了来自 NVIDIA 的专有内核驱动程序。然而，上游 Linux 内核社区接受了一项更改，此项更改将专有内核驱动程序（如 NVIDIA GPU 驱动程序）隔离开来，使之无法与其它内核驱动程序通信。此更改禁用了 P4 和 P5 系列实例上的 GPUDirect RDMA，这是一种允许高效使用 EFA GPUs 进行分布式训练的机制。因此， DLAMIs 现在使用 OpenRM 驱动程序（NVIDIA 开源驱动程序），该驱动程序与开源 EFA 驱动程序链接以支持 g4dn、G5、P4 和 P5。但是，此 OpenRM 驱动程序不支持较旧的实例（例如 P3 和 G3）。因此，为了确保我们继续提供支持这两种实例类型的最新、高性能和安全性 DLAMIs ，我们 DLAMIs 分为两组：一组使用 OpenRM 驱动程序（支持 G4dn、G5、P4 和 P5），另一组使用较旧的专有驱动程序（支持 P3、p3dn 和 G3）。

这一变化影响 DLAMIs 了哪些？

这一变化影响了所有人 DLAMIs。

这对您意味着什么？

只要您在支持的亚马逊弹性计算云 (Amazon EC2) 实例类型上运行，它们都 DLAMIs 将继续提供功能、性能和安全性。要确定 DLAMI 支持的 EC2 实例类型，请查看该 DLAMI 的发行说明，然后查找支持的实例。 EC2 有关当前支持的 DLAMI 选项的列表及指向其发布说明的链接，请参阅 的发行说明 DLAMIs。

此外，您必须使用正确的 Amazon Command Line Interface (Amazon CLI) 命令来调用当前 DLAMIs。

对于支持 P3、p3dn 和 G3 的基础 DLAMIs ，请使用以下命令：

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base Proprietary Nvidia Driver AMI (Amazon
 Linux 2) Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

对于支持 g4dN、G5、P4 和 P5 的基础 DLAMIs ，请使用以下命令：

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base OSS Nvidia Driver AMI (Amazon Linux 2)
 Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

较新的版本会失去功能 DLAMIs吗？

否，不损失任何功能。当前版本 DLAMIs 提供前一个版本的所有功能、性能和安全性 DLAMIs，前提是你在支持的 EC2 实例类型上运行它们。

这一变化是否影响了 Deep Learning Containers？

不，此更改并未影响 Dee Amazon p Learning Containers，因为它们不包括 NVIDIA 驱动程序。但是，请务必在与底层实例兼容的深度学习容器上 AMIs 运行 Deep Learning Containers。

有关 DLAMI 的相关信息

可以在《 Amazon Deep Learning AMIs 开发人员指南》之外找到其它资源，其中包含有关 DLAMI 的相关信息。在 Amazon Web Services re:Post，查看其他客户提出的有关DLAMI的问题，或者自己提问。在 Mach Amazon ine Learning Amazon 博客和其他博客上，阅读有关 DLAMI 的官方文章。

Amazon Web Services re:Post

标签： Amazon Deep Learning AMIs

Amazon 博客

- Amazon Machine Learning 博客 | 分类： Amazon Deep Learning AMIs
- Amazon Machine Learning 博客 | 在 Amazon EC2 C5 和 P3 实例上优化了 TensorFlow 1.6，训练速度更快
- Amazon Machine Learning 博客 | Amazon Deep Learning AMIs 面向机器学习从业者的新内容
- Amazon Partner Network (APN) 博客 | 推出新的培训课程：Machine Learning 和深度学习简介 Amazon
- Amazon 新闻博客 | 深度学习之旅 Amazon

DLAMI 的已弃用功能

下表列出了 Amazon Deep Learning AMIs (DLAMI) 的已弃用功能、我们弃用这些功能的日期以及我们为何弃用这些功能的详细信息。

功能	日期	详细信息
Ubuntu 16.04	10/07/2021	Ubuntu Linux 16.04 LTS 于 2021 年 4 月 30 日结束了为期五年的 LTS 窗口，其供应商不再提供支持。自 2021 年 10 月起，不再在新版本中更新深度学习基础 AMI (Ubuntu 16.04)。先前的版本将继续可用。
Amazon Linux	10/07/2021	截至 2020 年 12 月 end-of-life，亚马逊 Linux 已上市。自 2021 年10 月起，不再在新版本中更新深度学习 AMI (Amazon Linux)。先前的深度学习 AMI (Amazon Linux) 版本将继续可用。
Chainer	2020 年 7 月 1 日	Chainer 宣布自 2019 年 12 月起停用主要版本。因此，从 2020 年 7 月起，我们将不再在 DLAMI 中包含 Chainer Conda 环境。包含这些环境的先前 DLAMI 版本将继续可用。仅在开源社区针对这些框架发布安全修补程序时，我们才会为这些环境提供更新。

功能	日期	详细信息
Python 3.6	2020 年 6 月 15 日	由于客户的要求，我们正在迁移到 Python 3.7 来TF/MX/PT发布新版本。
Python 2	2020 年 1 月 1 日	Python 开源社区已正式结束对 Python 2 的支持。 TensorFlow PyTorch、和 MXNet 社区还宣布， TensorFlow 1.15、TensorFlow 2.1、1. PyTorch 4 和 MXNet 1.6.0 版本将是最后一个支持 Python 2 的版本。

DLAMI 的文档历史记录

下表提供了最近 DLAMI 版本以及对《 Amazon Deep Learning AMIs 开发人员指南》的相关更改的历史记录。

最近更改

变更	说明	日期
使用 TensorFlow 服务来训练 MNIST 模型	使用 Tensorflow 服务来训练 MNIST 模型的示例。	2025 年 2 月 14 日
ARM64 DLAMI	Amazon Deep Learning AMIs 现在支持基于 Arm64 处理器的 GPUs图像。	2021 年 11 月 29 日
TensorFlow 2	带有 Conda 的 Deep Learning AMI 现在有 TensorFlow 2 个，带有 CUDA 10。	2019 年 12 月 3 日
Amazon 推论	深度学习 AMI 现在支持 Amazon Inferentia 硬件和 Amazon Neuron SDK。	2019 年 12 月 3 日
PyTorch 从夜间版本中安装	添加了一个教程，介绍如何使用 Conda 卸载 PyTorch，然后在深度学习 AMI PyTorch 上安装每晚版本。	2018 年 9 月 25 日
Conda 教程	已更新示例 MOTD 以反映更新的版本。	2018 年 7 月 23 日

之前的更改

下表提供了 2018 年 7 月之前早期 DLAMI 版本和相关更改的历史记录。

更改	描述	日期
TensorFlow 和 Horovod	添加了 ImageNet 使用 TensorFlow 和 Horovod 进行训练的教程。	2018 年 6 月 6 日
升级指南	添加了升级指南。	2018 年 5 月 15 日
新区域和新的 10 分钟教程	添加的新区域：美国西部 (加利福利亚北部)、南美洲、加拿大 (中部)、欧洲 (伦敦) 和欧洲 (巴黎)。此外，首次发布的 10 分钟教程的标题为：“深度学习 AMI 入门”。	2018 年 4 月 26 日
Chainer 教程	添加了有关在多 GPU、单个 GPU 和 CPU 中使用 Chainer 的教程。CUDA 集成已针对多个框架从 CUDA 8 升级到 CUDA 9。	2018 年 2 月 28 日
Linux AMIs v3.0，以及 MXNet 模型服务器、 TensorFlow 服务和 TensorBoard	添加了 Conda 的教程， AMIs 其中包含使用模型服务器 v0.1.5、Serving v1.4.0 和 v0.4.0 的新 MXNet 模型和可视化 TensorFlow 服务功能。 TensorBoard AMI 和框架 CUDA 功能（在 Conda 和 CUDA 概述中描述）。最新发行说明迁移到 https://www.amazonaws.cn/releasenotes/	2018 年 1 月 25 日
Linux AMIs v2.0	Base、Source 和 Conda AMIs 更新了 NCCL 2.1。Source 和 Conda AMIs 更新了 MXNet	2017 年 12 月 11 日

更改	描述	日期
	v1.0、 PyTorch 0.3.0 和 Keras 2.0.9。	
添加了两个 Windows AMI 选项	Windows 2012 R2 和 2016 已 AMIs 发布：已添加到 AMI 选择指南中并添加到发行说明中。	2017 年 11 月 30 日
初始文档版本	更改的详细说明，带有指向所更改主题/章节的链接。	2017 年 11 月 15 日

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