Developer Guide

AWS Deep Learning AMIs

AWS Deep Learning AMIs: Developer Guide

Table of Contents

What is the DLAMI? 1
About this guide 1
Prerequisites 1
Example use cases 1
Features 2
Preinstalled frameworks 2
Preinstalled GPU software 3
Model serving and visualization 3
Release notes for DLAMIs 4
Base DLAMIs 4
Single-framework DLAMIs 5
Multi-framework DLAMIs 6
Getting started 7
Choosing a DLAMI 7
CUDA Installations and Framework Bindings 8
Base 9
Conda 9
Architecture 11
OS 11
Choosing an instance 11
Pricing 13
Region Availability 13
GPU 13
CPU 14
Inferentia 15
Trainium 16
Setting up 17
Finding a DLAMI ID 17
Launching an instance 19
Connecting to an instance 21
Setting up Jupyter 21
Securing server 22
Starting server 23
Connecting client 23

Logging in ... 25
Cleaning up ... 26
Using a DLAMI ... 28
Conda DLAMI ... 28
Introduction to the Deep Learning AMI with Conda ... 28
Log in to Your DLAMI ... 29
Start the TensorFlow Environment ... 29
Switch to the PyTorch Python 3 Environment ... 30
Removing Environments ... 31
Base DLAMI ... 31
Using the Deep Learning Base AMI ... 31
Configuring CUDA Versions ... 32
Jupyter Notebooks ... 32
Navigating the Installed Tutorials ... 33
Switching Environments with Jupyter ... 33
Tutorials ... 34
Activating Frameworks ... 34
Elastic Fabric Adapter ... 37
GPU Monitoring and Optimization ... 52
AWS Inferentia ... 62
ARM64 DLAMI ... 84
Inference ... 87
Model Serving ... 87
Upgrading Your DLAMI ... 94
DLAMI Upgrade ... 94
Software Updates ... 95
Release Notifications ... 96
Security ... 98
Data protection ... 99
Identity and access management ... 100
Authenticating with identities ... 100
Managing access using policies ... 103
IAM with Amazon EMR ... 105
Compliance validation ... 106
Resilience ... 106
Infrastructure security ... 107

Monitoring ... 107
Usage tracking ... 107
Framework support policy ... 109
DLAMI framework support FAQs ... 109
What framework versions get security patches? ... 110
What images does AWS publish when new framework versions are released? ... 110
What images get new SageMaker AI/AWS features? ... 110
How is current version defined in the Supported Frameworks table? ... 110
What if I am running a version that is not in the Supported Frameworks table? ... 110
Do DLAMIs support previous versions of TensorFlow? ... 110
How can I find the latest patched image for a supported framework version? ... 111
How frequently are new images released? ... 111
Will my instance be patched in place while my workload is running? ... 111
What happens when a new patched or updated framework version is available? ... 111
Are dependencies updated without changing the framework version? ... 112
When does active support for my framework version end? ... 112
Will images with framework versions that are no longer actively maintained be patched? ... 113
How do I use an older framework version? ... 113
How do I stay up-to-date with support changes in frameworks and their versions? ... 114
Do I need a commercial license to use the Anaconda Repository? ... 114
Important changes ... 115
DLAMI NVIDIA driver change FAQs ... 115
What changed? ... 115
Why was this change required? ... 116
Which DLAMIs did this change affect? ... 116
What does this mean for you? ... 117
Is there any loss of functionality with the newer DLAMIs? ... 117
Did this change affect Deep Learning Containers? ... 117
Related information ... 118
Deprecated features ... 119
Document history ... 121

What is AWS Deep Learning AMIs?

AWS Deep Learning AMIs (DLAMI) provides customized machine images that you can use for deep learning in the cloud. The DLAMIs are available in most AWS Regions for a variety of Amazon Elastic Compute Cloud (Amazon EC2) instance types, from a small CPU-only instance to the latest high-powered multi-GPU instances. The DLAMIs come preconfigured with NVIDIA CUDA and NVIDIA cuDNN and the latest releases of the most popular deep learning frameworks.

About this guide

The content in the can help you launch and use the DLAMIs. The guide covers several common deep learning use cases, for both training and inference. It also covers how to choose the right AMI for your purpose and the kind of instances that you might prefer.

Additionally, the DLAMIs include several tutorials that their supported frameworks provide. This guide can show you how to activate each framework and find the appropriate tutorials to get started. It also has tutorials on distributed training, debugging, using AWS Inferentia and AWS Trainium, and other key concepts. For instructions on how to set up a Jupyter notebook server to run the tutorials in your browser, see Setting up a Jupyter Notebook server on a DLAMI instance.

Prerequisites

To successfully run the DLAMIs, we recommend that you be familiar with command line tools and basic Python.

Example DLAMI use cases

The following are examples of some common use cases for AWS Deep Learning AMIs (DLAMI).

Learning about deep learning – DLAMI is a great choice for learning or teaching machine learning and deep learning frameworks. The DLAMIs take away the headache from troubleshooting the installations of each framework and getting them to play along on the same computer. The DLAMIs include a Jupyter notebook and make it easy to run the tutorials that the frameworks provide for people new to machine learning and deep learning.

App development – If you're an app developer who's interested in using deep learning to make your apps utilize the latest advances in AI, then DLAMI is the perfect test bed for you. Each framework comes with tutorials on how to get started with deep learning, and many of them have model zoos that make it easy to try out deep learning without having to create the neural networks yourself or to do any of the model training. Some examples show you how to build an image detection application in just a few minutes, or how to build a speech recognition app for your own chatbot.

Machine learning and data analytics – If you're a data scientist or you're interested in processing your data with deep learning, then you'll find that many of the frameworks have support for R and Spark. You will find tutorials on how to do simple regressions, all the way up to building scalable data processing systems for personalization and predictions systems.

Research – If you're a researcher who wants to try out a new framework, test a new model, or train new models, then DLAMI and AWS capabilities for scale can alleviate the pain of tedious installations and management of multiple training nodes.

Note

While your initial choice might be to upgrade your instance type to a larger instance with more GPUs (up to 8), you can also scale horizontally by creating a cluster of DLAMI instances. Check out Related information about DLAMI for more information on cluster builds.

Features of DLAMI

The features of AWS Deep Learning AMIs (DLAMI) include preinstalled deep learning frameworks, GPU software, model servers, and model visualization tools.

Preinstalled frameworks

There are currently two primary flavors of DLAMI with other variations related to the operating system (OS) and software versions:

- Deep Learning AMI with Conda – Frameworks installed separately using conda packages and separate Python environments.
- Deep Learning Base AMI – No frameworks installed; only NVIDIA CUDA and other dependencies.

The Deep Learning AMI with Conda uses conda environments to isolate each framework, so you can switch between them at will and not worry about their dependencies conflicting. The Deep Learning AMI with Conda supports the following frameworks:

- PyTorch
- TensorFlow 2

Note

DLAMI no longer supports the following deep learning frameworks: Apache MXNet, Microsoft Cognitive Toolkit (CNTK), Caffe, Caffe2, Theano, Chainer, and Keras.

Preinstalled GPU software

Even if you use a CPU-only instance, the DLAMIs will have NVIDIA CUDA and NVIDIA cuDNN. The installed software is the same regardless of the instance type. Keep in mind that GPU-specific tools work only on an instance that has at least one GPU. For more information about instance types, see Choosing a DLAMI instance type.

For more information about CUDA, see CUDA Installations and Framework Bindings.

Model serving and visualization

Deep Learning AMI with Conda comes preinstalled with model servers for TensorFlow, as well as TensorBoard for model visualizations. For more information, see TensorFlow Serving.

Release notes for DLAMIs

Here you can find detailed release notes for all currently supported AWS Deep Learning AMIs (DLAMI) options.

For release notes for DLAMI frameworks that we no longer support, see the **Unsupported Framework Release Notes Archive** section of the DLAMI Framework Support Policy page.

Note

The AWS Deep Learning AMIs have a nightly release cadence for security patches. We don't include these incremental security patches in official release notes.

Base DLAMIs

GPU

- **X86**
  - AWS Deep Learning Base AMI (Amazon Linux 2023)
  - AWS Deep Learning Base AMI (Ubuntu 22.04)
  - AWS Deep Learning Base AMI (Ubuntu 20.04)
  - AWS Deep Learning Base AMI (Amazon Linux 2)
- **ARM64**
  - AWS Deep Learning Base ARM64 AMI (Ubuntu 22.04)
  - AWS Deep Learning Base ARM64 AMI (Amazon Linux 2)

AWS Neuron

- **X86**
  - AWS Deep Learning Base AMI Neuron (Amazon Linux 2)
  - AWS Deep Learning Base AMI Neuron (Ubuntu 20.04)

Qualcomm

- **X86**
  - AWS Deep Learning Base Qualcomm AMI (Amazon Linux 2)

Single-framework DLAMIs

PyTorch-specific AMIs

GPU

- **X86**
  - AWS Deep Learning AMI GPU PyTorch 2.5 (Ubuntu 22.04)
  - AWS Deep Learning AMI GPU PyTorch 2.5 (Amazon Linux 2023)
  - AWS Deep Learning AMI GPU PyTorch 2.4 (Ubuntu 22.04)
  - AWS Deep Learning AMI GPU PyTorch 2.3 (Ubuntu 20.04)
  - AWS Deep Learning AMI GPU PyTorch 2.3 (Amazon Linux 2)
  - AWS Deep Learning AMI GPU PyTorch 2.2 (Ubuntu 20.04)
  - AWS Deep Learning AMI GPU PyTorch 2.2 (Amazon Linux 2)
- **ARM64**
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.5 (Ubuntu 22.04)
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.4 (Ubuntu 22.04)
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.3 (Ubuntu 22.04)
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.2 (Ubuntu 20.04)

AWS Neuron

- **X86**
  - AWS Deep Learning AMI Neuron PyTorch 1.13 (Amazon Linux 2)
  - AWS Deep Learning AMI Neuron PyTorch 1.13 (Ubuntu 20.04)

TensorFlow-specific AMIs

GPU

- **X86**

  - AWS Deep Learning AMI GPU TensorFlow 2.18 (Amazon Linux 2023)
  - AWS Deep Learning AMI GPU TensorFlow 2.18 (Ubuntu 22.04)
  - AWS Deep Learning AMI GPU TensorFlow 2.17 (Ubuntu 22.04)
  - AWS Deep Learning AMI GPU TensorFlow 2.16 (Amazon Linux 2)
  - AWS Deep Learning AMI GPU TensorFlow 2.16 (Ubuntu 20.04)

AWS Neuron

- **X86**
  - AWS Deep Learning AMI Neuron TensorFlow 2.10 (Amazon Linux 2)
  - AWS Deep Learning AMI Neuron TensorFlow 2.10 (Ubuntu 20.04)

Multi-framework DLAMIs

Tip

If you use only one machine learning framework, then we recommend a single-framework DLAMI.

GPU

- **X86**
  - AWS Deep Learning AMI (Amazon Linux 2)

AWS Neuron

- **X86**
  - AWS Deep Learning AMI Neuron (Amazon Linux 2023)
  - AWS Deep Learning AMI Neuron (Ubuntu 22.04)

Getting started with DLAMI

This guide includes tips about picking the DLAMI that's right for you, selecting an instance type that fits your use case and budget, and Related information about DLAMI that describes custom setups that may be of interest.

If you're new to using AWS or using Amazon EC2, start with the Deep Learning AMI with Conda. If you're familiar with Amazon EC2 and other AWS services like Amazon EMR, Amazon EFS, or Amazon S3, and are interested in integrating those services for projects that need distributed training or inference, then check out Related information about DLAMI to see if one fits your use case.

We recommend that you check out Choosing a DLAMI to get an idea of which instance type might be best for your application.

Next step

Choosing a DLAMI

Choosing a DLAMI

We offer a range of DLAMI options as mentioned in the GPU DLAMI release notes. To help you select the correct DLAMI for your use case, we group images by the hardware type or functionality for which they were developed. Our top level groupings are:

- **DLAMI Type:** Base, Single-Framework, Multi-Framework (Conda DLAMI)
- **Compute Architecture:** x86-based, Arm64-based AWS Graviton
- **Processor Type:** GPU, CPU, Inferentia, Trainium
- **SDK:** CUDA, AWS Neuron
- **OS:** Amazon Linux, Ubuntu

The rest of the topics in this guide help further inform you and go into more details.

Topics

- CUDA Installations and Framework Bindings
- Deep Learning Base AMI

- Deep Learning AMI with Conda
- DLAMI Architecture Options
- DLAMI Operating System Options

Next Up

Deep Learning AMI with Conda

CUDA Installations and Framework Bindings

While deep learning is all pretty cutting edge, each framework offers "stable" versions. These stable versions may not work with the latest CUDA or cuDNN implementation and features. Your use case and the features you require can help you choose a framework. If you are not sure, then use the latest Deep Learning AMI with Conda. It has official `pip` binaries for all frameworks with CUDA, using whichever most recent version is supported by each framework. If you want the latest versions, and to customize your deep learning environment, use the Deep Learning Base AMI.

Look at our guide on Stable Versus Release Candidates for further guidance.

Choose a DLAMI with CUDA

The Deep Learning Base AMI has all available CUDA version series

The Deep Learning AMI with Conda has all available CUDA version series

Note

We no longer include the MXNet, CNTK, Caffe, Caffe2, Theano, Chainer, or Keras Conda environments in the AWS Deep Learning AMIs.

For specific framework version numbers, see the Release notes for DLAMIs

Choose this DLAMI type or learn more about the different DLAMIs with the **Next Up** option.

Choose one of the CUDA versions and review the full list of DLAMIs that have that version in the **Appendix**, or learn more about the different DLAMIs with the **Next Up** option.

Next Up

Deep Learning Base AMI

Related Topics

- For instructions on switching between CUDA versions, refer to the Using the Deep Learning Base AMI tutorial.

Deep Learning Base AMI

The Deep Learning Base AMI is like an empty canvas for deep learning. It comes with everything you need up until the point of the installation of a particular framework, and has your choice of CUDA versions.

Why to Choose the Base DLAMI

This AMI group is useful for project contributors who want to fork a deep learning project and build the latest. It's for someone who wants to roll their own environment with the confidence that the latest NVIDIA software is installed and working so they can focus on picking which frameworks and versions they want to install.

Choose this DLAMI type or learn more about the different DLAMIs with the **Next Up** option.

Next Up

DLAMI with Conda

Related Topics

- Using the Deep Learning Base AMI

Deep Learning AMI with Conda

The Conda DLAMI uses `conda` virtual environments, they are present either multi-framework or single framework DLAMIs. These environments are configured to keep the different framework installations separate and streamline switching between frameworks. This is great for learning and experimenting with all of the frameworks the DLAMI has to offer. Most users find that the new Deep Learning AMI with Conda is perfect for them.

They are updated often with the latest versions from the frameworks, and have the latest GPU drivers and software. They are generally referred to as *the* AWS Deep Learning AMIs in most documents. These DLAMIs support Ubuntu 20.04, Ubuntu 22.04, Amazon Linux 2, Amazon Linux 2023 Operating systems. Operating systems support depends on support from upstream OS.

Stable Versus Release Candidates

The Conda AMIs use optimized binaries of the most recent formal releases from each framework. Release candidates and experimental features are not to be expected. The optimizations depend on the framework's support for acceleration technologies like Intel's MKL DNN, which speeds up training and inference on C5 and C4 CPU instance types. The binaries are also compiled to support advanced Intel instruction sets including but not limited to AVX, AVX-2, SSE4.1, and SSE4.2. These accelerate vector and floating point operations on Intel CPU architectures. Additionally, for GPU instance types, the CUDA and cuDNN are updated with whichever version the latest official release supports.

The Deep Learning AMI with Conda automatically installs the most optimized version of the framework for your Amazon EC2 instance upon the framework's first activation. For more information, refer to Using the Deep Learning AMI with Conda.

If you want to install from source, using custom or optimized build options, the Deep Learning Base AMIs might be a better option for you.

Python 2 Deprecation

The Python open source community has officially ended support for Python 2 on January 1, 2020. The TensorFlow and PyTorch community have announced that the TensorFlow 2.1 and PyTorch 1.4 releases are the last ones supporting Python 2. Previous releases of the DLAMI (v26, v25, etc) that contain Python 2 Conda environments continue to be available. However, we provide updates to the Python 2 Conda environments on previously published DLAMI versions only if there are security fixes published by the open-source community for those versions. DLAMI releases with the latest versions of the TensorFlow and PyTorch frameworks do not contain the Python 2 Conda environments.

CUDA Support

Specific CUDA version numbers can be found in the GPU DLAMI release notes.

Next Up

DLAMI Architecture Options

Related Topics

- For a tutorial on using a Deep Learning AMI with Conda, see the Using the Deep Learning AMI with Conda tutorial.

DLAMI Architecture Options

AWS Deep Learning AMIss are offered with either x86-based or Arm64-based AWS Graviton2 architectures.

For information about getting started with the ARM64 GPU DLAMI, see The ARM64 DLAMI. For more details on available instance types, see Choosing a DLAMI instance type.

Next Up

DLAMI Operating System Options

DLAMI Operating System Options

DLAMIs are offered in the following operating systems.

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

Older versions of operating systems are available on deprecated DLAMIs. For more information on DLAMI deprecation, see Deprecations for DLAMI

Before choosing a DLAMI, assess what instance type you need and identify your AWS Region.

Next Up

Choosing a DLAMI instance type

Choosing a DLAMI instance type

More generally, consider the following when choosing an instance type for a DLAMI.

- If you're new to deep learning, then an instance with a single GPU might suit your needs.
- If you're budget conscious, then you can use CPU-only instances.
- If you're looking to optimize high performance and cost efficiency for deep learning model inference, then you can use instances with AWS Inferentia chips.
- If you're looking for a high performance GPU instance with an Arm64-based CPU architecture, then you can use the G5g instance type.
- If you're interested in running a pretrained model for inference and predictions, then you can attach an Amazon Elastic Inference to your Amazon EC2 instance. Amazon Elastic Inference gives you access to an accelerator with a fraction of a GPU.
- For high-volume inference services, a single CPU instance with a lot of memory, or a cluster of such instances, might be a better solution.
- If you're using a large model with a lot of data or a high batch size, then you need a larger instance with more memory. You can also distribute your model to a cluster of GPUs. You may find that using an instance with less memory is a better solution for you if you decrease your batch size. This may impact your accuracy and training speed.
- If you’re interested in running machine learning applications using NVIDIA Collective Communications Library (NCCL) requiring high levels of inter-node communications at scale, you might want to use Elastic Fabric Adapter (EFA).

For more detail on instances, see EC2 Instance Types.

The following topics provide information about instance type considerations.

Important

The Deep Learning AMIs include drivers, software, or toolkits developed, owned, or provided by NVIDIA Corporation. You agree to use these NVIDIA drivers, software, or toolkits only on Amazon EC2 instances that include NVIDIA hardware.

Topics

- Pricing for the DLAMI
- DLAMI Region Availability
- Recommended GPU Instances

- Recommended CPU Instances
- Recommended Inferentia Instances
- Recommended Trainium Instances

Pricing for the DLAMI

The deep learning frameworks included in the DLAMI are free, and each has its own open-source licenses. Although the software included in the DLAMI is free, you still have to pay for the underlying Amazon EC2 instance hardware.

Some Amazon EC2 instance types are labeled as free. It is possible to run the DLAMI on one of these free instances. This means that using the DLAMI is entirely free when you only use that instance's capacity. If you need a more powerful instance with more CPU cores, more disk space, more RAM, or one or more GPUs, then you need an instance that is not in the free-tier instance class.

For more information about instance choice and pricing, see Amazon EC2 pricing.

DLAMI Region Availability

Each Region supports a different range of instance types and often an instance type has a slightly different cost in different Regions. DLAMIs are not available in every Region, but it is possible to copy DLAMIs to the Region of your choice. See Copying an AMI for more information. Note the Region selection list and be sure you pick a Region that's close to you or your customers. If you plan to use more than one DLAMI and potentially create a cluster, be sure to use the same Region for all of nodes in the cluster.

For a more info on Regions, visit Amazon EC2 service endpoints.

Next Up

Recommended GPU Instances

Recommended GPU Instances

We recommend a GPU instance for most deep learning purposes. Training new models is faster on a GPU instance than a CPU instance. You can scale sub-linearly when you have multi-GPU instances or if you use distributed training across many instances with GPUs.

The following instance types support the DLAMI. For information about GPU instance type options and their uses, see EC2 Instance Types and select **Accelerated Computing**.

Note

The size of your model should be a factor in choosing an instance. If your model exceeds an instance's available RAM, choose a different instance type with enough memory for your application.

- Amazon EC2 P5e Instances have up to 8 NVIDIA Tesla H200 GPUs.
- Amazon EC2 P5 Instances have up to 8 NVIDIA Tesla H100 GPUs.
- Amazon EC2 P4 Instances have up to 8 NVIDIA Tesla A100 GPUs.
- Amazon EC2 P3 Instances have up to 8 NVIDIA Tesla V100 GPUs.
- Amazon EC2 G3 Instances have up to 4 NVIDIA Tesla M60 GPUs.
- Amazon EC2 G4 Instances have up to 4 NVIDIA T4 GPUs.
- Amazon EC2 G5 Instances have up to 8 NVIDIA A10G GPUs.
- Amazon EC2 G6 Instances have up to 8 NVIDIA L4 GPUs.
- Amazon EC2 G6e Instances have up to 8 NVIDIA L40S Tensor Core GPUs.
- Amazon EC2 G5g Instances have Arm64-based AWS Graviton2 processors.

DLAMI instances provide tooling to monitor and optimize your GPU processes. For more information about monitoring your GPU processes, see GPU Monitoring and Optimization.

For specific tutorials on working with G5g instances, see The ARM64 DLAMI.

Next Up

Recommended CPU Instances

Recommended CPU Instances

Whether you're on a budget, learning about deep learning, or just want to run a prediction service, you have many affordable options in the CPU category. Some frameworks take advantage of Intel's MKL DNN, which speeds up training and inference on C5 (not available in all Regions) CPU instance types. For information about CPU instance types, see EC2 Instance Types and select **Compute Optimized**.

Note

The size of your model should be a factor in choosing an instance. If your model exceeds an instance's available RAM, choose a different instance type with enough memory for your application.

- Amazon EC2 C5 Instances have up to 72 Intel vCPUs. C5 instances excel at scientific modeling, batch processing, distributed analytics, high-performance computing (HPC), and machine and deep learning inference.

Next Up

Recommended Inferentia Instances

Recommended Inferentia Instances

AWS Inferentia instances are designed to provide high performance and cost efficiency for deep learning model inference workloads. Specifically, Inf2 instance types use AWS Inferentia chips and the AWS Neuron SDK, which is integrated with popular machine learning frameworks such as TensorFlow and PyTorch.

Customers can use Inf2 instances to run large scale machine learning inference applications such as search, recommendation engines, computer vision, speech recognition, natural language processing, personalization, and fraud detection, at the lowest cost in the cloud.

Note

The size of your model should be a factor in choosing an instance. If your model exceeds an instance's available RAM, choose a different instance type with enough memory for your application.

- Amazon EC2 Inf2 Instances have up to up to 16 AWS Inferentia chips and 100 Gbps of networking throughput.

For more information about getting started with AWS Inferentia DLAMIs, see The AWS Inferentia Chip With DLAMI.

Next Up

Recommended Trainium Instances

Recommended Trainium Instances

AWS Trainium instances are designed to provide high performance and cost efficiency for deep learning model inference workloads. Specifically, Trn1 instance types use AWS Trainium chips and the AWS Neuron SDK, which is integrated with popular machine learning frameworks such as TensorFlow and PyTorch.

Customers can use Trn1 instances to run large scale machine learning inference applications such as search, recommendation engines, computer vision, speech recognition, natural language processing, personalization, and fraud detection, at the lowest cost in the cloud.

Note

The size of your model should be a factor in choosing an instance. If your model exceeds an instance's available RAM, choose a different instance type with enough memory for your application.

- Amazon EC2 Trn1 Instances have up to up to 16 AWS Trainium chips and 100 Gbps of networking throughput.

Setting up a DLAMI instance

After you choose a DLAMI and choose an Amazon Elastic Compute Cloud (Amazon EC2) instance type that you want to use, then you're ready to set up your new DLAMI instance.

If you haven't yet chosen a DLAMI and EC2 instance type, then see Getting started with DLAMI.

Topics

- Finding the ID of a DLAMI
- Launching a DLAMI instance
- Connecting to a DLAMI instance
- Setting up a Jupyter Notebook server on a DLAMI instance
- Cleaning up a DLAMI instance

Finding the ID of a DLAMI

Each DLAMI has a unique identifier (ID). When you launch a DLAMI instance using the Amazon EC2 console, you can optionally use the DLAMI ID to search for the DLAMI that you want to use. When you launch a DLAMI instance using the AWS Command Line Interface (AWS CLI), this ID is required.

You can find the ID for the DLAMI of your choice by using an AWS CLI command for either Amazon EC2 or Parameter Store, a capability of AWS Systems Manager. For instructions on installing and configuring the AWS CLI, see Get started with the AWS CLI in the *AWS Command Line Interface User Guide*.

Using Parameter Store

To find a DLAMI ID using ssm get-parameter

In the following **ssm get-parameter** command, for the `--name` option, the parameter name format is `/aws/service/deeplearning/ami/$architecture/$ami_type/latest/ami-id`. In this name format, *`architecture`* can be either **`x86_64`** or **`arm64`**. Specify the *`ami_type`* by taking the DLAMI name and removing the keywords "deep", "learning", and "ami". AMI Name can be found in Release notes for DLAMIs.

Important

To use this command, the AWS Identity and Access Management (IAM) principal that you use must have the `ssm:GetParameter` permission. For more information about IAM principals, see the Additional resources section of **IAM roles** in the *IAM User Guide*.

-
  ```
  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
  ```

  The output should look similar to the following:

  ```
  ami-09ee1a996ac214ce7
  ```

  Tip

  For some currently supported DLAMI frameworks, you can find more specific example **ssm get-parameter** commands in Release notes for DLAMIs. Choose the link to the release notes of your chosen DLAMI, and then find its ID query in the release notes.

Using Amazon EC2 CLI

To find a DLAMI ID using ec2 describe-images

In the following **ec2 describe-images** command, for the value of the filter `Name=name`, enter the DLAMI name. You can specify a release version for a given framework, or you can get the latest release by replacing the version number with a question mark (?).

-
  ```
  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
  ```

  The output should look similar to the following:

```
ami-09ee1a996ac214ce7
```

> **Tip**
>
> For an example **ec2 describe-images** command that's specific to the DLAMI of your choice, see Release notes for DLAMIs. Choose the link to the release notes of your chosen DLAMI, and then find its ID query in the release notes.

Next step

Launching a DLAMI instance

Launching a DLAMI instance

After you find the ID of the DLAMI that you want to use to launch a DLAMI instance, you're ready to launch the instance. To launch it, you can use either the Amazon EC2 console or the AWS Command Line Interface (AWS CLI).

> **Note**
>
> For this walkthrough, we might make references specific to the Deep Learning Base OSS Nvidia Driver GPU AMI (Ubuntu 22.04). Even if you select a different DLAMI, you should be able to follow this guide.

EC2 console

> **Note**
>
> To accelerate high-performance computing (HPC) and machine learning applications, you can launch your DLAMI instance with an Elastic Fabric Adapter (EFA). For specific instructions, see Launching a AWS Deep Learning AMIs Instance With EFA.

1. Open the EC2 Console.

2. Note your current AWS Region in the top-most navigation. If this isn't your desired Region, then change this option before you continue. For more information, see Amazon EC2 service endpoints in the *Amazon Web Services General Reference*.
3. Choose **Launch Instance**.
4. Enter a name for your instance and select the DLAMI that is right for you.

    a. Find an existing DLAMI in **My AMIs** or choose **Quick Start.**

    b. Search by DLAMI ID. Browse the options then select your choice.
5. Choose an instance type. You can find the recommended instance families for your DLAMI in Release notes for DLAMIs. For general recommendations on DLAMI instance types, see Choosing a DLAMI instance type.
6. Choose **Launch Instance**.

AWS CLI

- To use the AWS CLI, you must have the ID of the DLAMI that you want to use, the AWS Region and EC2 instance type, and your security token information. Then, you can launch the instance using the **ec2 run-instances** AWS CLI command.

  For instructions on installing and configuring the AWS CLI, see Get started with the AWS CLI in the *AWS Command Line Interface User Guide*. For more information, including command examples, see Launch, list, and close Amazon EC2 instances for the AWS CLI.

After you launch your instance using either the Amazon EC2 console or AWS CLI, wait for the instance to be ready. This usually takes only a few minutes. You can verify the status of the instance in the Amazon EC2 console. For more information, see Status checks for Amazon EC2 instances in the *Amazon EC2 User Guide*.

Next step

Connecting to a DLAMI instance

Connecting to a DLAMI instance

After you launch a DLAMI instance and the instance is running, you can connect to it from a client (Windows, macOS, or Linux) using SSH. For instructions, see Connect to your Linux instance using SSH in the *Amazon EC2 User Guide*.

Keep a copy of the SSH login command handy in case you want to set up a Jupyter Notebook server after you log in. To connect to the Jupyter webpage, you use a variation of that command.

Next step

Setting up a Jupyter Notebook server on a DLAMI instance

Setting up a Jupyter Notebook server on a DLAMI instance

With a Jupyter Notebook server, you can create and run Jupyter notebooks from your DLAMI instance. With Jupyter notebooks, you can conduct machine learning (ML) experiments for training and inference while using the AWS infrastructure and accessing packages built into the DLAMI. For more information about Jupyter notebooks, see The Jupyter Notebook on the Jupyter User Documentation website.

To set up a Jupyter Notebook server, you must:

- Configure the Jupyter Notebook server on your DLAMI instance.
- Configure your client to connect to the Jupyter Notebook server. We provide configuration instructions for Windows, macOS, and Linux clients.
- Test the setup by logging in to the Jupyter Notebook server.

To complete these steps, follow the instructions in the following topics. After you've set up a Jupyter Notebook server, you can run the example notebook tutorials that ship in the DLAMIs. For more information, see Running Jupyter Notebook Tutorials.

Topics

- Securing the Jupyter Notebook server on a DLAMI instance
- Starting the Jupyter Notebook server on a DLAMI instance
- Connecting a client to the Jupyter Notebook server on a DLAMI instance
- Logging in to the Jupyter Notebook server on a DLAMI instance

Securing the Jupyter Notebook server on a DLAMI instance

To keep your Jupyter Notebook server secure, we recommend setting up a password and creating an SSL certificate for the server. To configure a password and SSL, first connect to your DLAMI instance, and then follow these instructions.

To secure the Jupyter Notebook server

1. Jupyter provides a password utility. Run the following command and enter your preferred password at the prompt.

   ```
   $ jupyter notebook password
   ```

   The output will look something like this:

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

2. Create a self-signed SSL certificate. Follow the prompts to fill out your locality as you see fit. You must enter `.` if you wish to leave a prompt blank. Your answers will not impact the functionality of the certificate.

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

> **Note**
>
> You might be interested in creating a regular SSL certificate that is third-party signed and does not cause the browser to give you a security warning. This process is much more involved. For more information, see Securing a notebook server in the Jupyter Notebook user documentation.

Next step

Starting the Jupyter Notebook server on a DLAMI instance

Starting the Jupyter Notebook server on a DLAMI instance

After you secure your Jupyter Notebook server with a password and SSL, you can start the server. Log in to your DLAMI instance and run the following command that uses the SSL certificate that you created previously.

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

With the server started, you can now connect to it via an SSH tunnel from your client computer. When the server runs, you will see some output from Jupyter confirming that the server is running. At this point, ignore the callout that you can access the server via a local host URL, because that won't work until you create the tunnel.

Note

Jupyter will handle switching environments for you when you switch frameworks using the Jupyter web interface. For more information, see Switching Environments with Jupyter.

Next step

Connecting a client to the Jupyter Notebook server on a DLAMI instance

Connecting a client to the Jupyter Notebook server on a DLAMI instance

After you start the Jupyter Notebook server on your DLAMI instance, configure your Windows, macOS, or Linux client to connect to the server. When you connect, you can create and access Jupyter notebooks on the server in your workspace and run your deep learning code on the server.

Prerequisites

Be sure you have the following, which you need to set up an SSH tunnel:

- The public DNS name of your Amazon EC2 instance. For more information, see Amazon EC2 instance hostname types in the *Amazon EC2 User Guide*.

- The key pair for the private key file. For more information about accessing your key pair, see Amazon EC2 key pairs and Amazon EC2 instances in the *Amazon EC2 User Guide*.

Connect from a Windows, macOS, or Linux client

To connect to your DLAMI instance from a Windows, macOS, or Linux client, follow the instructions for your client's operating system.

Windows

To connect to your DLAMI instance from a Windows client using SSH

1. Use an SSH client for Windows, such as PuTTY. For instructions, see Connect to your Linux instance using PuTTY in the *Amazon EC2 User Guide*. For other SSH connection options, see Connect to your Linux instance using SSH.
2. (Optional) Create an SSH tunnel to a running Jupyter server. Install Git Bash on your Windows client, and then follow the connection instructions for macOS and Linux clients.

macOS or Linux

To connect to your DLAMI instance from a macOS or Linux client using SSH

1. Open a terminal.
2. Run the following command to forward all requests on local port 8888 to port 8888 on your remote Amazon EC2 instance. Update the command by replacing the location of your key to access the Amazon EC2 instance and the public DNS name of your Amazon EC2 instance. Note, for an Amazon Linux AMI, the user name is `ec2-user` instead of `ubuntu`.

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

   This command opens a tunnel between your client and the remote Amazon EC2 instance that is running the Jupyter Notebook server.

Next step

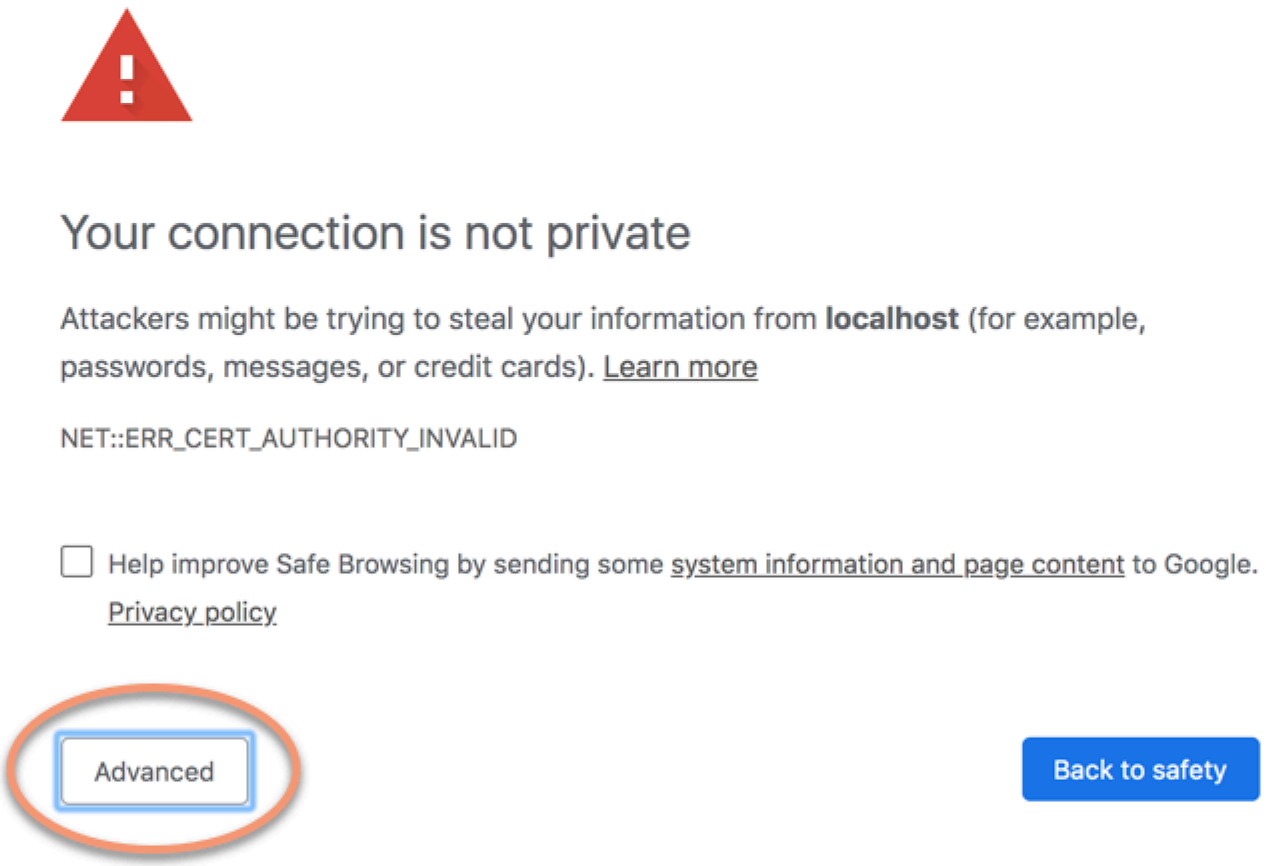
Logging in to the Jupyter Notebook server on a DLAMI instance

Logging in to the Jupyter Notebook server on a DLAMI instance

After you connect your client to the Jupyter Notebook server on your DLAMI instance, you can log in to the server.

To log in to the server in your browser

1. In the address bar of your browser, enter the following URL, or click on this link: https://localhost:8888
2. With a self-signed SSL certificate, your browser will warn you and prompt you to avoid continuing to visit the website.



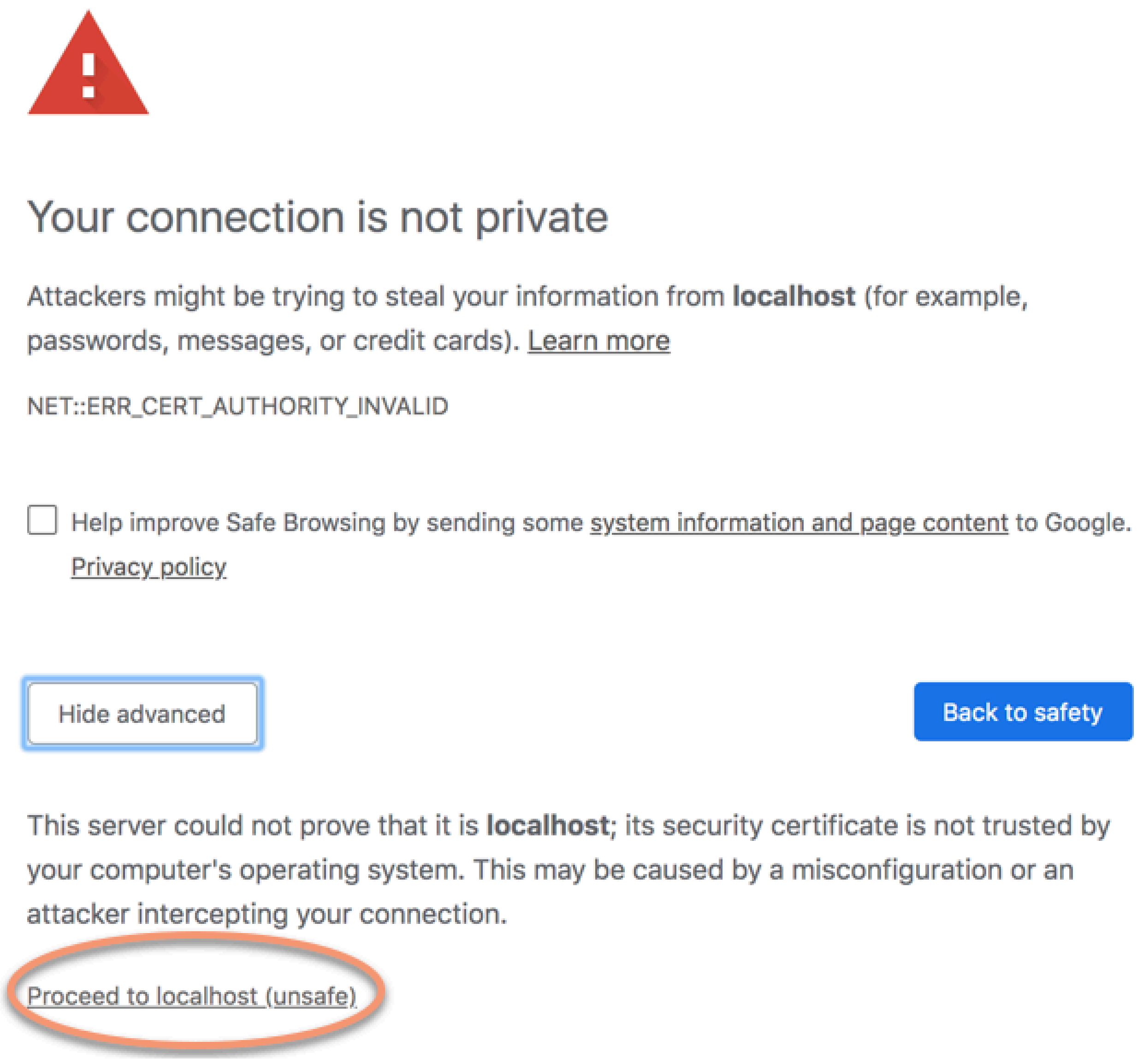
Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

☐ Help improve Safe Browsing by sending some system information and page content to Google. Privacy policy

Since you set this up yourself, it is safe to continue. Depending your browser you will get an "advanced", "show details", or similar button.



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

☐ Help improve Safe Browsing by sending some system information and page content to Google. Privacy policy

This server could not prove that it is **localhost**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to localhost (unsafe)

Click on this, then click on the "proceed to localhost" link. If the connection is successful, you see the Jupyter Notebook server webpage. At this point, you will be asked for the password you previously set up.

Now you have access to the Jupyter Notebook server that is running on the DLAMI instance. You can create new notebooks or run the provided Tutorials.

Cleaning up a DLAMI instance

When you no longer need your DLAMI instance, you can stop it or terminate it on Amazon EC2 to avoid incurring unexpected charges.

If you stop an instance, you can keep it around and start it later when you want to use it again. Your configurations, files, and other non-volatile information are stored in a volume on Amazon Simple Storage Service (Amazon S3). While your instance is stopped, you incur S3 charges for retaining the volume, but you don't incur charges for compute resources. When your start the instance again, it will mount that storage volume with your data.

If you terminate an instance, it's gone, and you cannot start it again. Of course, you won't incur any more charges for the compute resources with a terminated instance. However, your data still resides on Amazon S3, and you can continue to incur S3 charges. To prevent all further charges related to your terminated instance, you must also delete the storage volume on Amazon S3. For instructions, see Terminate Amazon EC2 instances in the *Amazon EC2 User Guide*.

For more information about Amazon EC2 instance states, such as `stopped` and `terminated`, see Amazon EC2 instance state changes in the *Amazon EC2 User Guide*.

Using a DLAMI

Topics

- Using the Deep Learning AMI with Conda
- Using the Deep Learning Base AMI
- Running Jupyter Notebook Tutorials
- Tutorials

The following sections describe how the Deep Learning AMI with Conda can be used to switch environments, run sample code from each of the frameworks, and run Jupyter so you can try out different notebook tutorials.

Using the Deep Learning AMI with Conda

Topics

- Introduction to the Deep Learning AMI with Conda
- Log in to Your DLAMI
- Start the TensorFlow Environment
- Switch to the PyTorch Python 3 Environment
- Removing Environments

Introduction to the Deep Learning AMI with Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer.

The Deep Learning AMI with Conda has been configured for you to easily switch between deep learning environments. The following instructions guide you on some basic commands with `conda`. They also help you verify that the basic import of the framework is functioning, and that you can run a couple simple operations with the framework. You can then move on to more thorough tutorials provided with the DLAMI or the frameworks' examples found on each frameworks' project site.

Log in to Your DLAMI

After you log in to your server, you will see a server "message of the day" (MOTD) describing various Conda commands that you can use to switch between the different deep learning frameworks. Below is an example MOTD. Your specific MOTD may vary as new versions of the DLAMI are released.

```
=============================================================================
        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
    =============================================================================
```

Start the TensorFlow Environment

Note

When you launch your first Conda environment, please be patient while it loads. The Deep Learning AMI with Conda automatically installs the most optimized version of the framework for your EC2 instance upon the framework's first activation. You should not expect subsequent delays.

1. Activate the TensorFlow virtual environment for Python 3.

```
$ source activate tensorflow2_p310
```

2. Start the iPython terminal.

```
(tensorflow2_p310)$ ipython
```

3. Run a quick TensorFlow program.

```

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

You should see "Hello, Tensorflow!"

Next Up

Running Jupyter Notebook Tutorials

Switch to the PyTorch Python 3 Environment

If you're still in the iPython console, use `quit()`, then get ready to switch environments.

- Activate the PyTorch virtual environment for Python 3.

```
$ source activate pytorch_p310
```

Test Some PyTorch Code

To test your installation, use Python to write PyTorch code that creates and prints an array.

1. Start the iPython terminal.

```
(pytorch_p310)$ ipython
```

2. Import PyTorch.

```
import torch
```

You might see a warning message about a third-party package. You can ignore it.

3. Create a 5x3 matrix with the elements initialized randomly. Print the array.

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

Verify the result.

```
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]])
```

Removing Environments

If you run out of space on the DLAMI, you can choose to uninstall Conda packages that you are not using:

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

Using the Deep Learning Base AMI

Using the Deep Learning Base AMI

The Base AMI comes with a foundational platform of GPU drivers and acceleration libraries to deploy your own customized deep learning environment. By default the AMI is configured with any one NVIDIA CUDA version environment. You can also switch between different versions of CUDA. Refer to the following instructions for how to do this.

Configuring CUDA Versions

You can verify the CUDA version by running NVIDIA's `nvcc` program.

```
nvcc --version
```

You can select and verify a particular CUDA version with the following bash command:

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

For more information, see the Base DLAMI release notes.

Running Jupyter Notebook Tutorials

Tutorials and examples ship with each of the deep learning projects' source and in most cases they will run on any DLAMI. If you chose the Deep Learning AMI with Conda, you get the added benefit of a few hand-picked tutorials already set up and ready to try out.

Important

To run the Jupyter notebook tutorials installed on the DLAMI, you will need to Setting up a Jupyter Notebook server on a DLAMI instance.

Once the Jupyter server is running, you can run the tutorials through your web browser. If you are running the Deep Learning AMI with Conda or if you have set up Python environments, you can switch Python kernels from the Jupyter notebook interface. Select the appropriate kernel before trying to run a framework-specific tutorial. Further examples of this are provided for users of the Deep Learning AMI with Conda.

Note

Many tutorials require additional Python modules that may not be set up on your DLAMI. If you get an error like `"xyz module not found"`, log in to the DLAMI, activate the environment as described above, then install the necessary modules.

Tip

Deep learning tutorials and examples often rely on one or more GPUs. If your instance type doesn't have a GPU, you may need to change some of the example's code to get it to run.

Navigating the Installed Tutorials

Once you're logged in to the Jupyter server and can see the tutorials directory (on Deep Learning AMI with Conda only), you will be presented with folders of tutorials by each framework name. If you don't see a framework listed, then tutorials are not available for that framework on your current DLAMI. Click on the name of the framework to see the listed tutorials, then click a tutorial to launch it.

The first time you run a notebook on the Deep Learning AMI with Conda, it will want to know which environment you would like to use. It will prompt you to select from a list. Each environment is named according to this pattern:

```
Environment (conda_framework_python-version)
```

For example, you might see `Environment (conda_mxnet_p36)`, which signifies that the environment has MXNet and Python 3. The other variation of this would be `Environment (conda_mxnet_p27)`, which signifies that the environment has MXNet and Python 2.

Tip

If you're concerned about which version of CUDA is active, one way to see it is in the MOTD when you first log in to the DLAMI.

Switching Environments with Jupyter

If you decide to try a tutorial for a different framework, be sure to verify the currently running kernel. This info can be seen in the upper right of the Jupyter interface, just below the logout button. You can change the kernel on any open notebook by clicking the Jupyter menu item **Kernel**, then **Change Kernel**, and then clicking the environment that suits the notebook you're running.

At this point you'll need to rerun any cells because a change in the kernel will erase the state of anything you've run previously.

Tip

Switching between frameworks can be fun and educational, however you can run out of memory. If you start running into errors, look at the terminal window that has the Jupyter server running. There are helpful messages and error logging here, and you may see an out-of-memory error. To fix this, you can go to the home page of your Jupyter server, click the **Running** tab, then click **Shutdown** for each of the tutorials that are probably still running in the background and eating up all of your memory.

Tutorials

The following are tutorials on how to use the Deep Learning AMI with Conda's software.

Topics

- Activating Frameworks
- Distributed training using Elastic Fabric Adapter
- GPU Monitoring and Optimization
- The AWS Inferentia Chip With DLAMI
- The ARM64 DLAMI
- Inference
- Model Serving

Activating Frameworks

The following are the deep learning frameworks installed on the Deep Learning AMI with Conda. Click on a framework to learn how to activate it.

Topics

- PyTorch
- TensorFlow 2

PyTorch

Activating PyTorch

When a stable Conda package of a framework is released, it's tested and pre-installed on the DLAMI. If you want to run the latest, untested nightly build, you can Install PyTorch's Nightly Build (experimental) manually.

To activate the currently installed framework, follow these instructions on your Deep Learning AMI with Conda.

For PyTorch on Python 3 with CUDA and MKL-DNN, run this command:

```
$ source activate pytorch_p310
```

Start the iPython terminal.

```
(pytorch_p310)$ ipython
```

Run a quick PyTorch program.

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

You should see the initial random array printed, then its size, and then the addition of another random array.

Install PyTorch's Nightly Build (experimental)

How to install PyTorch from a nightly build

You can install the latest PyTorch build into either or both of the PyTorch Conda environments on your Deep Learning AMI with Conda.

1. • (Option for Python 3) - Activate the Python 3 PyTorch environment:

   ```
   $ source activate pytorch_p310
   ```

2. The remaining steps assume you are using the `pytorch_p310` environment. Remove the currently installed PyTorch:

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

3. - (Option for GPU instances) - Install the latest nightly build of PyTorch with CUDA.0:

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

   - (Option for CPU instances) - Install the latest nightly build of PyTorch for instances with no GPUs:

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

4. To verify you have successfully installed latest nightly build, start the IPython terminal and check the version of PyTorch.

   ```
   (pytorch_p310)$ ipython
   ```

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

   The output should print something similar to `1.0.0.dev20180922`

5. To verify that the PyTorch nightly build works well with the MNIST example, you can run a test script from PyTorch's examples repository:

   ```
   (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
   ```

More Tutorials

For further tutorials and examples refer to the framework's official docs, PyTorch documentation, and the PyTorch website.

TensorFlow 2

This tutorial shows how to activate TensorFlow 2 on an instance running the Deep Learning AMI with Conda (DLAMI on Conda) and run a TensorFlow 2 program.

When a stable Conda package of a framework is released, it's tested and pre-installed on the DLAMI.

Activating TensorFlow 2

To run TensorFlow on the DLAMI with Conda

1. To activate TensorFlow 2, open an Amazon Elastic Compute Cloud (Amazon EC2) instance of the DLAMI with Conda.
2. For TensorFlow 2 and Keras 2 on Python 3 with CUDA 10.1 and MKL-DNN, run this command:

```
$ source activate tensorflow2_p310
```

3. Start the iPython terminal:

```
(tensorflow2_p310)$ ipython
```

4. Run a TensorFlow 2 program to verify that it is working properly:

```


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

`Hello, TensorFlow!` should appear on your screen.

More Tutorials

For more tutorials and examples, see the TensorFlow documentation for the TensorFlow Python API or see the TensorFlow website.

Distributed training using Elastic Fabric Adapter

An Elastic Fabric Adapter (EFA) is a network device that you can attach to your DLAMI instance to accelerate High Performance Computing (HPC) applications. EFA enables you to achieve the application performance of an on-premises HPC cluster, with the scalability, flexibility, and elasticity provided by the AWS Cloud.

The following topics show you how to get started using EFA with the DLAMI.

Note

Choose your DLAMI from this Base GPU DLAMI list

Topics

- Launching a AWS Deep Learning AMIs Instance With EFA
- Using EFA on the DLAMI

Launching a AWS Deep Learning AMIs Instance With EFA

The latest Base DLAMI is ready to use with EFA and comes with the required drivers, kernel modules, libfabric, openmpi and the NCCL OFI plugin for GPU instances.

You can find the supported CUDA versions of a Base DLAMI in the release notes.

Note:

- When running a NCCL Application using `mpirun` on EFA, you will have to specify the full path to the EFA supported installation as:

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

- To enable your application to use EFA, add `FI_PROVIDER="efa"` to the `mpirun` command as shown in Using EFA on the DLAMI.

Topics

- Prepare an EFA Enabled Security Group
- Launch Your Instance
- Verify EFA Attachment

Prepare an EFA Enabled Security Group

EFA requires a security group that allows all inbound and outbound traffic to and from the security group itself. For more information, see the EFA Documentation.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Security Groups** and then choose **Create Security Group**.
3. In the **Create Security Group** window, do the following:
    - For **Security group name**, enter a descriptive name for the security group, such as `EFA-enabled security group`.
    - (Optional) For **Description**, enter a brief description of the security group.
    - For **VPC**, select the VPC into which you intend to launch your EFA-enabled instances.
    - Choose **Create**.
4. Select the security group that you created, and on the **Description** tab, copy the **Group ID**.
5. On the **Inbound** and **Outbound** tabs, do the following:
    - Choose **Edit**.
    - For **Type**, choose **All traffic**.
    - For **Source**, choose **Custom**.
    - Paste the security group ID that you copied into the field.
    - Choose **Save**.
6. Enable inbound traffic referring to Authorizing Inbound Traffic for Your Linux Instances. If you skip this step, you won't be able to communicate with your DLAMI instance.

Launch Your Instance

EFA on the AWS Deep Learning AMIs is currently supported with the following instance types and operating systems:

- P3dn: Amazon Linux 2, Ubuntu 20.04
- P4d, P4de: Amazon Linux 2, Amazon Linux 2023, Ubuntu 20.04, Ubuntu 22.04
- P5, P5e, P5en: Amazon Linux 2, Amazon Linux 2023, Ubuntu 20.04, Ubuntu 22.04

The following section shows how to launch an EFA enabled DLAMI instance. For more information on launching an EFA enabled instance, see Launch EFA-Enabled Instances into a Cluster Placement Group.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch Instance**.
3. On the **Choose an AMI** page, select a supported DLAMI found on the DLAMI Release Notes Page
4. On the **Choose an Instance Type** page, select one of the following supported instance types and then choose **Next: Configure Instance Details.** Refer to this link for the list of supported instances: Get started with EFA and MPI
5. On the **Configure Instance Details** page, do the following:

   - For **Number of instances**, enter the number of EFA-enabled instances that you want to launch.
   - For **Network** and **Subnet**, select the VPC and subnet into which to launch the instances.
   - [Optional] For **Placement group**, select **Add instance to placement group**. For best performance, launch the instances within a placement group.
   - [Optional] For **Placement group name**, select **Add to a new placement group**, enter a descriptive name for the placement group, and then for **Placement group strategy**, select **cluster**.
   - Make sure to enable the **"Elastic Fabric Adapter"** on this page. If this option is disabled, change the subnet to one that supports your selected instance type.
   - In the **Network Interfaces** section, for device **eth0**, choose **New network interface**. You can optionally specify a primary IPv4 address and one or more secondary IPv4 addresses. If you're launching the instance into a subnet that has an associated IPv6 CIDR block, you can optionally specify a primary IPv6 address and one or more secondary IPv6 addresses.
   - Choose **Next: Add Storage**.
6. On the **Add Storage** page, specify the volumes to attach to the instances in addition to the volumes specified by the AMI (such as the root device volume), and then choose **Next: Add Tags**.
7. On the **Add Tags** page, specify tags for the instances, such as a user-friendly name, and then choose **Next: Configure Security Group**.
8. On the **Configure Security Group** page, for **Assign a security group**, select **Select an existing security group**, and then select the security group that you created previously**.**

9. Choose **Review and Launch**.
10. On the **Review Instance Launch** page, review the settings, and then choose **Launch** to choose a key pair and to launch your instances.

Verify EFA Attachment

From the Console

After launching the instance, check the instance details in the AWS Console. To do this, select the instance in the EC2 console and look at the Description Tab in the lower pane on the page. Find the parameter ‘Network Interfaces: eth0’ and click on eth0 which opens a pop-up. Make sure that ‘Elastic Fabric Adapter’ is enabled.

If EFA is not enabled, you can fix this by either:

- Terminating the EC2 instance and launching a new one with the same steps. Make sure the EFA is attached.
- Attach EFA to an existing instance.

  1. In the EC2 Console, go to Network Interfaces.
  2. Click on Create a Network Interface.
  3. Select the same subnet that your instance is in.
  4. Make sure to enable the ‘Elastic Fabric Adapter’ and click on Create.
  5. Go back to the EC2 Instances Tab and select your instance.
  6. Go to Actions: Instance State and stop the instance before you attach EFA.
  7. From Actions, select Networking: Attach Network Interface.
  8. Select the interface you just created and click on attach.
  9. Restart your instance.

From the Instance

The following test script is already present on the DLAMI. Run it to ensure that the kernel modules are loaded correctly.

```
$ fi_info -p efa
```

Your output should look similar to the following.

```
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
```

Verify Security Group Configuration

The following test script is already present on the DLAMI. Run it to ensure that the security group you created is configured correctly.

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

Your output should look similar to the following.

```
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
```

If it stops responding or does not complete, ensure that your security group has the correct inbound/outbound rules.

Using EFA on the DLAMI

The following section describes how to use EFA to run multi-node applications on the AWS Deep Learning AMIs.

Running Multi-Node Applications with EFA

To run an application across a cluster of nodes the following configuration is required

Topics

- Enable Passwordless SSH
- Create Hosts File
- NCCL Tests

Enable Passwordless SSH

Select one node in your cluster as the leader node. The remaining nodes are referred to as the member nodes.

1. On the leader node, generate the RSA keypair.

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

2. Change the permissions of the private key on the leader node.

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

3. Copy the public key `~/.ssh/id_rsa.pub` to and append it to `~/.ssh/authorized_keys` of the member nodes in the cluster.
4. You should now be able to directly login to the member nodes from the leader node using the private ip.

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

5. Disable strictHostKeyChecking and enable agent forwarding on the leader node by adding the following to the ~/.ssh/config file on the leader node:

```
Host *
    ForwardAgent yes
```

```
Host *
    StrictHostKeyChecking no
```

6. On Amazon Linux 2 instances, run the following command on the leader node to provide correct permissions to the config file:

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

Create Hosts File

On the leader node, create a hosts file to identify the nodes in the cluster. The hosts file must have an entry for each node in the cluster. Create a file ~/hosts and add each node using the private ip as follows:

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

NCCL Tests

Note

These tests have been run using EFA version 1.30.0 and OFI NCCL Plugin 1.7.4.

Listed below are a subset of NCCL Tests provided by Nvidia to test both functionality and performance over multiple compute nodes

Supported Instances: P3dn, P4, P5

Functionality Tests

NCCL Message Transfer Multi Node Test

The nccl_message_transfer is a simple test to ensure that the NCCL OFI Plugin is working as expected. The test validates functionality of NCCL's connection establishment and data transfer APIs. Make sure you use the complete path to mpirun as shown in the example while running NCCL applications with EFA. Change the params `np` and `N` based on the number of instances and GPUs in your cluster. For more information, see the AWS OFI NCCL documentation.

The following nccl_message_transfer test is for a generic CUDA xx.x version. You can run the commands for any available CUDA version in your Amazon EC2 instance by replacing the CUDA version in the script.

```
$/opt/amazon/openmpi/bin/mpirun -n 2 -N 1 --hostfile hosts \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:$LD_LIBRARY_PATH \
--mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to none \
opt/aws-ofi-nccl/tests/nccl_message_transfer
```

Your output should look like the following. You can check the output to see that EFA is being used as the OFI provider.

```
INFO: Function: nccl_net_ofi_init Line: 1069: NET/OFI Selected Provider is efa (found 4
 nics)
INFO: Function: nccl_net_ofi_init Line: 1160: NET/OFI Using transport protocol SENDRECV
INFO: Function: configure_ep_inorder Line: 261: NET/OFI Setting
 FI_OPT_EFA_SENDRECV_IN_ORDER_ALIGNED_128_BYTES not supported.
INFO: Function: configure_nccl_proto Line: 227: NET/OFI Setting NCCL_PROTO to "simple"
INFO: Function: main Line: 86: NET/OFI Process rank 1 started. NCCLNet device used on
 ip-172-31-13-179 is AWS Libfabric.
INFO: Function: main Line: 91: NET/OFI Received 4 network devices
INFO: Function: main Line: 111: NET/OFI Network supports communication using CUDA
 buffers. Dev: 3
INFO: Function: main Line: 118: NET/OFI Server: Listening on dev 3
INFO: Function: main Line: 131: NET/OFI Send connection request to rank 1
INFO: Function: main Line: 173: NET/OFI Send connection request to rank 0
INFO: Function: main Line: 137: NET/OFI Server: Start accepting requests
INFO: Function: main Line: 141: NET/OFI Successfully accepted connection from rank 1
INFO: Function: main Line: 145: NET/OFI Send 8 requests to rank 1
INFO: Function: main Line: 179: NET/OFI Server: Start accepting requests
INFO: Function: main Line: 183: NET/OFI Successfully accepted connection from rank 0
INFO: Function: main Line: 187: NET/OFI Rank 1 posting 8 receive buffers
INFO: Function: main Line: 161: NET/OFI Successfully sent 8 requests to rank 1
INFO: Function: main Line: 251: NET/OFI Got completions for 8 requests for rank 0
INFO: Function: main Line: 251: NET/OFI Got completions for 8 requests for rank 1
```

Performance Tests

Multi-node NCCL Performance Test on P4d.24xlarge

To check NCCL Performance with EFA, run the standard NCCL Performance test that is available on the official NCCL-Tests Repo. The DLAMI comes with this test already built for CUDA XX.X. You can similarly run your own script with EFA.

When constructing your own script, refer to the following guidance:

- Use the complete path to mpirun as shown in the example while running NCCL applications with EFA.
- Change the params np and N based on the number of instances and GPUs in your cluster.
- Add the NCCL_DEBUG=INFO flag and make sure that the logs indicate EFA usage as "Selected Provider is EFA".
- Set the Training Log Location to parse for validation

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

Use the command `watch nvidia-smi` on any of the member nodes to monitor GPU usage. The following `watch nvidia-smi` commands are for a generic CUDA xx.x version and depend on the Operating System of your instance. You can run the commands for any available CUDA version in your Amazon EC2 instance by replacing the CUDA version in the script.

- Amazon Linux 2:

```
 $ /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:

```
$ /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}
```

Your output should look like the following:

```
# 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   9591 on ip-172-31-4-37 device  0 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  1 Group  0 Pid   9592 on ip-172-31-4-37 device  1 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  2 Group  0 Pid   9593 on ip-172-31-4-37 device  2 [0x20] NVIDIA A100-SXM4-40GB
#  Rank  3 Group  0 Pid   9594 on ip-172-31-4-37 device  3 [0x20] NVIDIA A100-SXM4-40GB
#  Rank  4 Group  0 Pid   9595 on ip-172-31-4-37 device  4 [0x90] NVIDIA A100-SXM4-40GB
#  Rank  5 Group  0 Pid   9596 on ip-172-31-4-37 device  5 [0x90] NVIDIA A100-SXM4-40GB
#  Rank  6 Group  0 Pid   9597 on ip-172-31-4-37 device  6 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank  7 Group  0 Pid   9598 on ip-172-31-4-37 device  7 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank  8 Group  0 Pid  10216 on ip-172-31-13-179 device  0 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  9 Group  0 Pid  10217 on ip-172-31-13-179 device  1 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank 10 Group  0 Pid  10218 on ip-172-31-13-179 device  2 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank 11 Group  0 Pid  10219 on ip-172-31-13-179 device  3 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank 12 Group  0 Pid  10220 on ip-172-31-13-179 device  4 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank 13 Group  0 Pid  10221 on ip-172-31-13-179 device  5 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank 14 Group  0 Pid  10222 on ip-172-31-13-179 device  6 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank 15 Group  0 Pid  10223 on ip-172-31-13-179 device  7 [0xa0] NVIDIA A100-
SXM4-40GB
ip-172-31-4-37:9591:9591 [0] NCCL INFO Bootstrap : Using ens32:172.31.4.37
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin_v6
 symbol.
```

```
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin
 symbol (v4 or v5).
ip-172-31-4-37:9591:9591 [0] NCCL INFO cudaDriverVersion 12020
NCCL version 2.18.5+cuda12.2
...
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.7.4-aws
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
 version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
 version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Running on p4d.24xlarge platform,
 Setting NCCL_TOPO_FILE environment variable to /opt/aws-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)
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.04    0.00    0.00      0
           0             0     float     sum      -1    11.01    0.00    0.00      0
 11.00    0.00    0.00      0
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.02    0.00    0.00      0
           0             0     float     sum      -1    11.01    0.00    0.00      0
 11.00    0.00    0.00      0
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.02    0.00    0.00      0
         256             4     float     sum      -1    632.7    0.00    0.00      0
 628.2    0.00    0.00      0
```

```
         512             8     float     sum      -1    627.4    0.00    0.00      0
 629.6    0.00    0.00      0
        1024            16     float     sum      -1    632.2    0.00    0.00      0
 631.7    0.00    0.00      0
        2048            32     float     sum      -1    631.0    0.00    0.00      0
 634.2    0.00    0.00      0
        4096            64     float     sum      -1    623.3    0.01    0.01      0
 633.6    0.01    0.01      0
        8192           128     float     sum      -1    635.1    0.01    0.01      0
 633.5    0.01    0.01      0
       16384           256     float     sum      -1    634.8    0.03    0.02      0
 637.0    0.03    0.02      0
       32768           512     float     sum      -1    647.9    0.05    0.05      0
 636.8    0.05    0.05      0
       65536          1024     float     sum      -1    658.9    0.10    0.09      0
 667.0    0.10    0.09      0
      131072          2048     float     sum      -1    671.9    0.20    0.18      0
 662.9    0.20    0.19      0
      262144          4096     float     sum      -1    692.1    0.38    0.36      0
 685.1    0.38    0.36      0
      524288          8192     float     sum      -1    715.3    0.73    0.69      0
 696.6    0.75    0.71      0
     1048576         16384     float     sum      -1    734.6    1.43    1.34      0
 729.2    1.44    1.35      0
     2097152         32768     float     sum      -1    785.9    2.67    2.50      0
 794.5    2.64    2.47      0
     4194304         65536     float     sum      -1    837.2    5.01    4.70      0
 837.6    5.01    4.69      0
     8388608        131072     float     sum      -1    929.2    9.03    8.46      0
 931.4    9.01    8.44      0
    16777216        262144     float     sum      -1   1773.6    9.46    8.87      0
 1772.8    9.46    8.87      0
    33554432        524288     float     sum      -1   2110.2   15.90   14.91      0
 2116.1   15.86   14.87      0
    67108864       1048576     float     sum      -1   2650.9   25.32   23.73      0
 2658.1   25.25   23.67      0
   134217728       2097152     float     sum      -1   3943.1   34.04   31.91      0
 3945.9   34.01   31.89      0
   268435456       4194304     float     sum      -1   7216.5   37.20   34.87      0
 7178.6   37.39   35.06      0
   536870912       8388608     float     sum      -1    13680   39.24   36.79      0
 13676   39.26   36.80      0
[  1073741824      16777216    float     sum      -1    25645   41.87   39.25      0
 25497   42.11   39.48      0 ] <- Used For Benchmark
```

```
...
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 7.46044
```

Validation Tests

To Validate that the EFA tests returned a valid result, please use the following tests to confirm:

- Get the instance type using EC2 Instance Metadata:

```
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)
```

- Run the Performance Tests
- Set the Following Parameters

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- Validate the Results as shown:

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

    # Information on how the version come from logs
    #
    # ip-172-31-27-205:6427:6427 [0] NCCL INFO cudaDriverVersion 12020
    # NCCL version 2.16.2+cuda11.8
    # ip-172-31-27-205:6427:6820 [0] NCCL INFO NET/OFI Initializing aws-ofi-nccl
 1.7.1-aws
    # ip-172-31-27-205:6427:6820 [0] NCCL INFO NET/OFI Using CUDA runtime version
 11060

    # cudaDriverVersion 12020  --> This is max supported cuda version by nvidia
 driver
    # NCCL version 2.16.2+cuda11.8 --> This is NCCL version compiled with cuda
 version
    # Using CUDA runtime version 11060 --> This is selected cuda version

    # Validation of logs
    grep "NET/OFI Using CUDA runtime version ${CUDA_RUNTIME_VERSION}" ${TRAINING_LOG}
 || { echo "Runtime cuda text not found"; exit 1; }
    grep "NET/OFI Initializing aws-ofi-nccl" ${TRAINING_LOG} || { echo "aws-ofi-nccl
 is not working, please check if it is installed correctly"; exit 1; }
    grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-
specific options text not found"; exit 1; }
    grep "Using network AWS Libfabric" ${TRAINING_LOG} || { echo "AWS Libfabric 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/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found:
 NET/AWS 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; }
        grep "aws-ofi-nccl/xml/p4d-24xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
 file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/AWS 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; }
        grep "aws-ofi-nccl/xml/p4de-24xl-topo.xml" ${TRAINING_LOG} || { echo
 "Topology file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
        grep "NET/AWS 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; }
        grep "aws-ofi-nccl/xml/p5.48xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
 file 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
```

- To access the benchmark data, we can parse the final row of table output from the Multi Node all_reduce test:

```
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 Monitoring and Optimization

The following section will guide you through GPU optimization and monitoring options. This section is organized like a typical workflow with monitoring overseeing preprocessing and training.

- Monitoring
  - Monitor GPUs with CloudWatch
- Optimization
  - Preprocessing
  - Training

Monitoring

Your DLAMI comes preinstalled with several GPU monitoring tools. This guide also mentions tools that are available to download and install.

- Monitor GPUs with CloudWatch - a preinstalled utility that reports GPU usage statistics to Amazon CloudWatch.
- nvidia-smi CLI - a utility to monitor overall GPU compute and memory utilization. This is preinstalled on your AWS Deep Learning AMIs (DLAMI).
- NVML C library - a C-based API to directly access GPU monitoring and management functions. This used by the nvidia-smi CLI under the hood and is preinstalled on your DLAMI. It also has

Python and Perl bindings to facilitate development in those languages. The gpumon.py utility preinstalled on your DLAMI uses the pynvml package from nvidia-ml-py.

- NVIDIA DCGM - A cluster management tool. Visit the developer page to learn how to install and configure this tool.

Tip

Check out NVIDIA's developer blog for the latest info on using the CUDA tools installed your DLAMI:

- Monitoring TensorCore utilization using Nsight IDE and nvprof.

Monitor GPUs with CloudWatch

When you use your DLAMI with a GPU you might find that you are looking for ways to track its usage during training or inference. This can be useful for optimizing your data pipeline, and tuning your deep learning network.

There are two ways to configure GPU metrics with CloudWatch:

- Configure metrics with the AWS CloudWatch agent (Recommended)
- Configure metrics with the preinstalled `gpumon.py` script

Configure metrics with the AWS CloudWatch agent (Recommended)

Integrate your DLAMI with the unified CloudWatch agent to configure GPU metrics and monitor the utilization of GPU coprocesses in Amazon EC2 accelerated instances.

There are four ways to configure GPU metrics with your DLAMI:

- Configure minimal GPU metrics
- Configure partial GPU metrics
- Configure all available GPU metrics
- Configure custom GPU metrics

For information on updates and security patches, see Security patching for the AWS CloudWatch agent

Prerequisites

To get started, you must configure Amazon EC2 instance IAM permissions that allow your instance to push metrics to CloudWatch. For detailed steps, see Create IAM roles and users for use with the CloudWatch agent.

Configure minimal GPU metrics

Configure minimal GPU metrics using the `dlami-cloudwatch-agent@minimal` `systemd` service. This service configures the following metrics:

- `utilization_gpu`
- `utilization_memory`

You can find the `systemd` service for minimal preconfigured GPU metrics in the following location:

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

Enable and start the `systemd` service with the following commands:

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

Configure partial GPU metrics

Configure partial GPU metrics using the `dlami-cloudwatch-agent@partial` `systemd` service. This service configures the following metrics:

- `utilization_gpu`
- `utilization_memory`
- `memory_total`
- `memory_used`
- `memory_free`

You can find the `systemd` service for partial preconfigured GPU metrics in the following location:

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

Enable and start the `systemd` service with the following commands:

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

Configure all available GPU metrics

Configure all available GPU metrics using the `dlami-cloudwatch-agent@all` `systemd` service. This service configures the following metrics:

- `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`

You can find the `systemd` service for all available preconfigured GPU metrics in the following location:

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

Enable and start the `systemd` service with the following commands:

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

Configure custom GPU metrics

If the preconfigured metrics do not meet your requirements, you can create a custom CloudWatch agent configuration file.

Create a custom configuration file

To create a custom configuration file, refer to the detailed steps in [Manually create or edit the CloudWatch agent configuration file].

For this example, assume that the schema definition is located at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

Configure metrics with your custom file

Run the following command to configure the CloudWatch agent according to your custom file:

```
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
```

Security patching for the AWS CloudWatch agent

Newly released DLAMIs are configured with the latest available AWS CloudWatch agent security patches. Refer to the following sections to update your current DLAMI with the latest security patches depending on your operating system of choice.

Amazon Linux 2

Use `yum` to get the latest AWS CloudWatch agent security patches for an Amazon Linux 2 DLAMI.

```
 sudo yum update
```

Ubuntu

To get the latest AWS CloudWatch security patches for a DLAMI with Ubuntu, it is necessary to reinstall the AWS CloudWatch agent using an Amazon S3 download link.

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

For more information on installing the AWS CloudWatch agent using Amazon S3 download links, see Installing and running the CloudWatch agent on your servers.

Configure metrics with the preinstalled gpumon.py script

A utility called gpumon.py is preinstalled on your DLAMI. It integrates with CloudWatch and supports monitoring of per-GPU usage: GPU memory, GPU temperature, and GPU Power. The script periodically sends the monitored data to CloudWatch. You can configure the level of granularity for data being sent to CloudWatch by changing a few settings in the script. Before starting the script, however, you will need to setup CloudWatch to receive the metrics.

How to setup and run GPU monitoring with CloudWatch

1. Create an IAM user, or modify an existing one to have a policy for publishing the metric to CloudWatch. If you create a new user please take note of the credentials as you will need these in the next step.

   The IAM policy to search for is “cloudwatch:PutMetricData”. The policy that is added is as follows:

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

Tip

For more information on creating an IAM user and adding policies for CloudWatch, refer to the CloudWatch documentation.

2. On your DLAMI, run AWS configure and specify the IAM user credentials.

```
$ aws configure
```

3. You might need to make some modifications to the gpumon utility before you run it. You can find the gpumon utility and README in the location defined in the following code block. For more information on the `gpumon.py` script, see the Amazon S3 location of the script.

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

   Options:

   - Change the region in gpumon.py if your instance is NOT in us-east-1.
   - Change other parameters such as the CloudWatch `namespace` or the reporting period with `store_reso`.

4. Currently the script only supports Python 3. Activate your preferred framework's Python 3 environment or activate the DLAMI general Python 3 environment.

```
$ source activate python3
```

5. Run the gpumon utility in background.

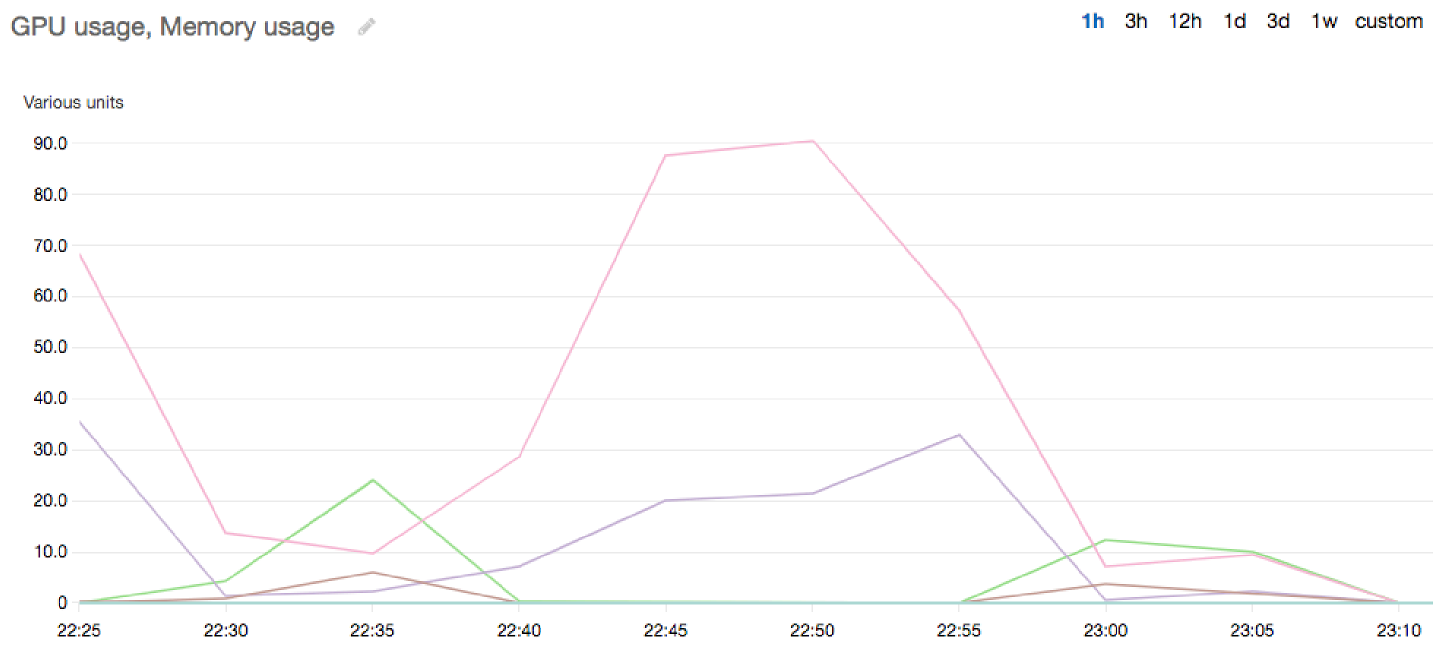
```
(python3)$ python gpumon.py &
```

6. Open your browser to the https://console.aws.amazon.com/cloudwatch/ then select metric. It will have a namespace 'DeepLearningTrain'.

Tip

You can change the namespace by modifying gpumon.py. You can also modify the reporting interval by adjusting `store_reso`.

The following is an example CloudWatch chart reporting on a run of gpumon.py monitoring a training job on p2.8xlarge instance.



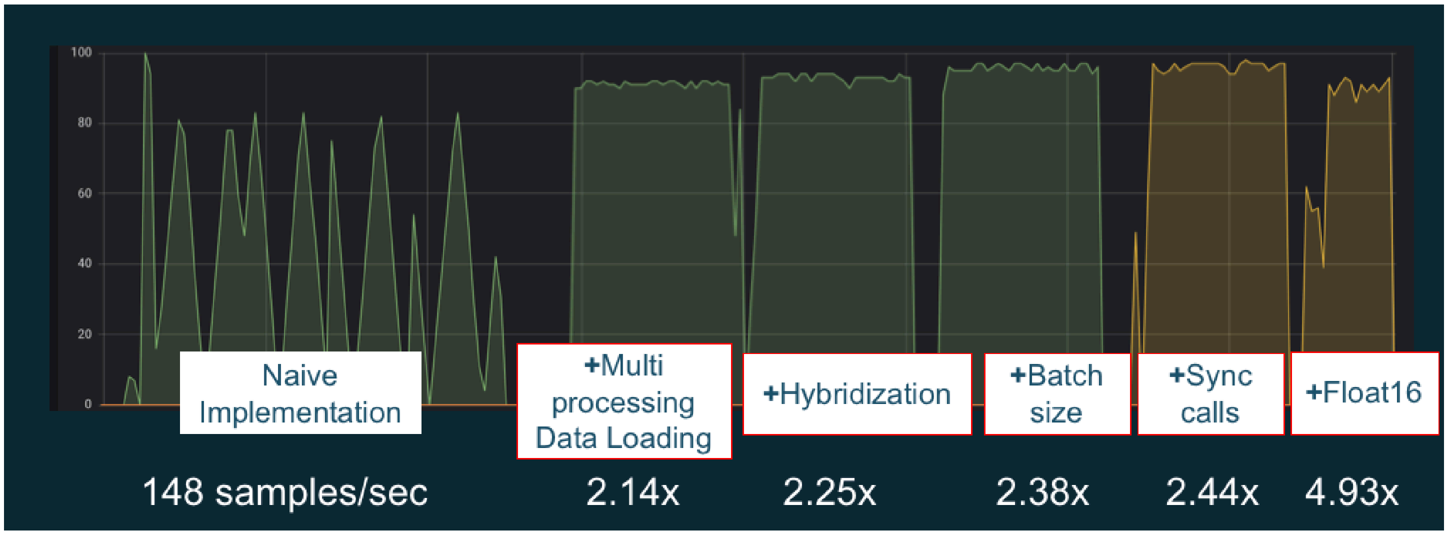
You might be interested in these other topics on GPU monitoring and optimization:

- Monitoring
  - Monitor GPUs with CloudWatch
- Optimization
  - Preprocessing
  - Training

Optimization

To make the most of your GPUs, you can optimize your data pipeline and tune your deep learning network. As the following chart describes, a naive or basic implementation of a neural network might use the GPU inconsistently and not to its fullest potential. When you optimize your preprocessing and data loading, you can reduce the bottleneck from your CPU to your GPU. You can adjust the neural network itself, by using hybridization (when supported by the framework), adjusting batch size, and synchronizing calls. You can also use multiple-precision (float16 or int8) training in most frameworks, which can have a dramatic effect on improving throughput.

The following chart shows the cumulative performance gains when applying different optimizations. Your results will depend on the data you are processing and the network you are optimizing.



Example GPU performance optimizations. Chart source: Performance Tricks with MXNet Gluon

The following guides introduce options that will work with your DLAMI and help you boost GPU performance.

Topics

- Preprocessing
- Training

Preprocessing

Data preprocessing through transformations or augmentations can often be a CPU-bound process, and this can be the bottleneck in your overall pipeline. Frameworks have built-in operators for image processing, but DALI (Data Augmentation Library) demonstrates improved performance over frameworks' built-in options.

- NVIDIA Data Augmentation Library (DALI): DALI offloads data augmentation to the GPU. It is not preinstalled on the DLAMI, but you can access it by installing it or loading a supported framework container on your DLAMI or other Amazon Elastic Compute Cloud instance. Refer to the DALI project page on the NVIDIA website for details. For an example use-case and to download code samples, see the SageMaker Preprocessing Training Performance sample.

- nvJPEG: a GPU-accelerated JPEG decoder library for C programmers. It supports decoding single images or batches as well as subsequent transformation operations that are common in deep learning. nvJPEG comes built-in with DALI, or you can download from the NVIDIA website's nvjpeg page and use it separately.

You might be interested in these other topics on GPU monitoring and optimization:

- Monitoring
  - Monitor GPUs with CloudWatch
- Optimization
  - Preprocessing
  - Training

Training

With mixed-precision training you can deploy larger networks with the same amount of memory, or reduce memory usage compared to your single or double precision network, and you will see compute performance increases. You also get the benefit of smaller and faster data transfers, an important factor in multiple node distributed training. To take advantage of mixed-precision training you need to adjust data casting and loss scaling. The following are guides describing how to do this for the frameworks that support mixed-precision.

- NVIDIA Deep Learning SDK - docs on the NVIDIA website describing mixed-precision implementation for MXNet, PyTorch, and TensorFlow.

Tip

Be sure to check the website for your framework of choice, and search for "mixed precision" or "fp16" for the latest optimization techniques. Here are some mixed-precision guides you might find helpful:

- Mixed-precision training with TensorFlow (video) - on the NVIDIA blog site.

- Mixed-precision training using float16 with MXNet - an FAQ article on the MXNet website.
- NVIDIA Apex: a tool for easy mixed-precision training with PyTorch - a blog article on the NVIDIA website.

You might be interested in these other topics on GPU monitoring and optimization:

- Monitoring
  - Monitor GPUs with CloudWatch
- Optimization
  - Preprocessing
  - Training

The AWS Inferentia Chip With DLAMI

AWS Inferentia is a custom machine learning chip designed by AWS that you can use for high-performance inference predictions. In order to use the chip, set up an Amazon Elastic Compute Cloud instance and use the AWS Neuron software development kit (SDK) to invoke the Inferentia chip. To provide customers with the best Inferentia experience, Neuron has been built into the AWS Deep Learning AMIs (DLAMI).

The following topics show you how to get started using Inferentia with the DLAMI.

Contents

- Launching a DLAMI Instance with AWS Neuron
- Using the DLAMI with AWS Neuron

Launching a DLAMI Instance with AWS Neuron

The latest DLAMI is ready to use with AWS Inferentia and comes with the AWS Neuron API package. To launch a DLAMI instance, see Launching and Configuring a DLAMI. After you have a DLAMI, use the steps here to ensure that your AWS Inferentia chip and AWS Neuron resources are active.

Contents

- Verify Your Instance

- Identifying AWS Inferentia Devices
- View Resource Usage
- Using Neuron Monitor (neuron-monitor)
- Upgrading Neuron Software

Verify Your Instance

Before using your instance, verify that it's properly setup and configured with Neuron.

Identifying AWS Inferentia Devices

To identify the number of Inferentia devices on your instance, use the following command:

```
neuron-ls
```

If your instance has Inferentia devices attached to it, your output will look similar to the following:

```
+--------+--------+--------+-----------+--------------+
| 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 |
+--------+--------+--------+-----------+--------------+
```

The supplied output is taken from an Inf1.6xlarge instance and includes the following columns:

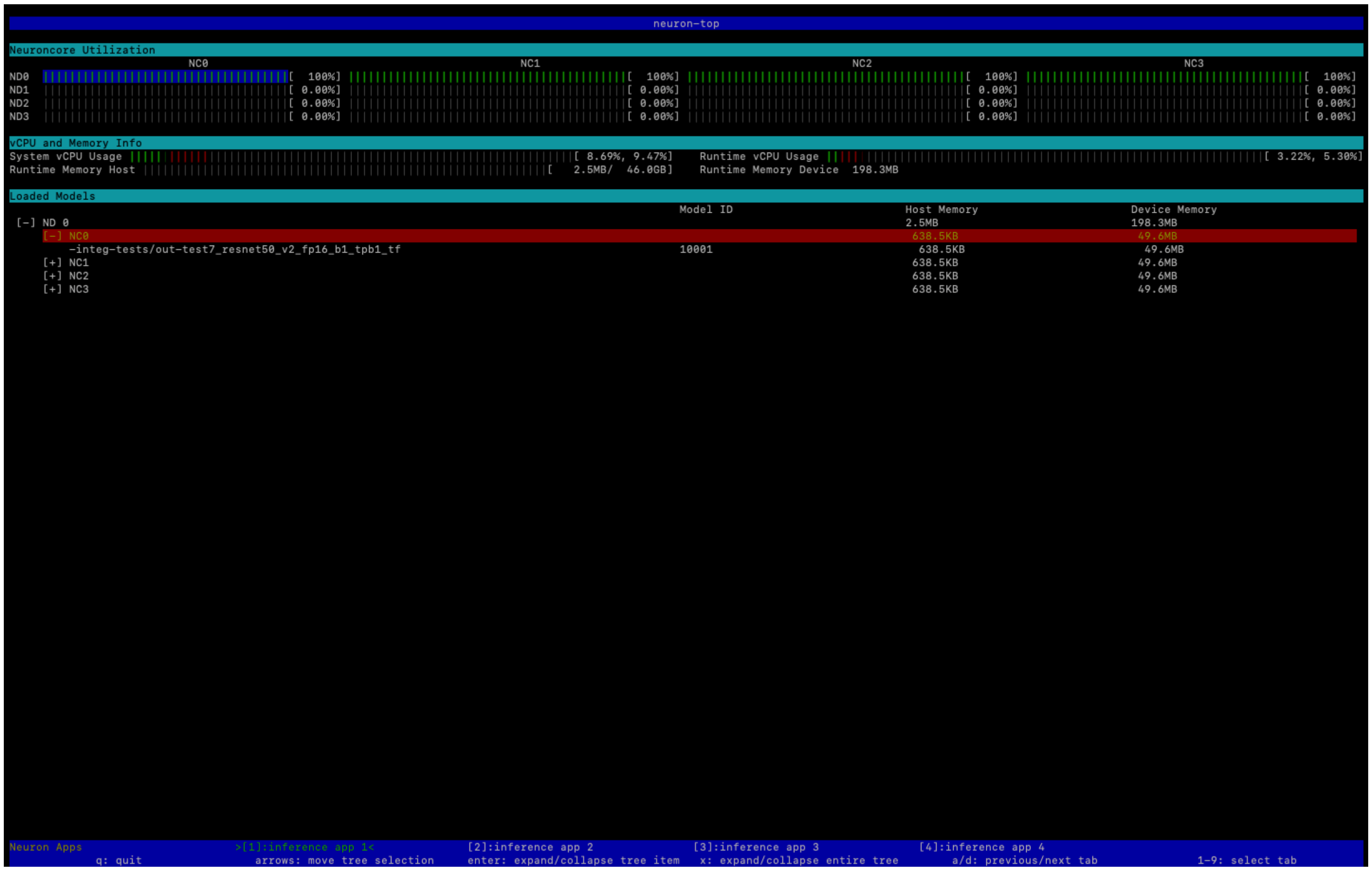
- NEURON DEVICE: The logical ID assigned to the NeuronDevice. This ID is used when configuring multiple runtimes to use different NeuronDevices.
- NEURON CORES: The number of NeuronCores present in the NeuronDevice.
- NEURON MEMORY: The amount of DRAM memory in the NeuronDevice.
- CONNECTED DEVICES: Other NeuronDevices connected to the NeuronDevice.
- PCI BDF: The PCI Bus Device Function (BDF) ID of the NeuronDevice.

View Resource Usage

View useful information about NeuronCore and vCPU utilization, memory usage, loaded models, and Neuron applications with the `neuron-top` command. Launching `neuron-top` with no arguments will show data for all machine learning applications that utilize NeuronCores.

```
neuron-top
```

When an application is using four NeuronCores, the output should look similar to the following image:



```
                                              neuron-top
Neuroncore Utilization
                 NC0                               NC1                               NC2                               NC3
ND0 |||||||||||||||||||||||[  100%] ||||||||||||||||||||||||[  100%] ||||||||||||||||||||||||[  100%] ||||||||||||||||||||||||[  100%]
ND1 |||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%]
ND2 |||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%]
ND3 |||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%] ||||||||||||||||||||||||[ 0.00%]

vCPU and Memory Info
System vCPU Usage ||||||||||||||||||||||||||||[ 8.69%, 9.47%]    Runtime vCPU Usage ||||||||||||||||||||||||||||||||[ 3.22%, 5.30%]
Runtime Memory Host ||||||||||||||||||||||[   2.5MB/  46.0GB]    Runtime Memory Device  198.3MB

Loaded Models
                                                          Model ID                 Host Memory              Device Memory
 [-] ND 0                                                                          2.5MB                    198.3MB
     [-] NC0                                                                        638.5KB                  49.6MB
         -integ-tests/out-test7_resnet50_v2_fp16_b1_tpb1_tf       10001              638.5KB                  49.6MB
     [+] NC1                                                                        638.5KB                  49.6MB
     [+] NC2                                                                        638.5KB                  49.6MB
     [+] NC3                                                                        638.5KB                  49.6MB

Neuron Apps              >[1]:inference app 1<        [2]:inference app 2            [3]:inference app 3            [4]:inference app 4
            q: quit         arrows: move tree selection  enter: expand/collapse tree item  x: expand/collapse entire tree     a/d: previous/next tab          1-9: select tab
```

For more information on resources to monitor and optimize Neuron-based inference applications, see Neuron Tools.

Using Neuron Monitor (neuron-monitor)

Neuron Monitor collects metrics from the Neuron runtimes running on the system and streams the collected data to stdout in JSON format. These metrics are organized into metric groups that you configure by providing a configuration file. For more information on Neuron Monitor, see the User Guide for Neuron Monitor.

Upgrading Neuron Software

For information on how to update Neuron SDK software within DLAMI, see the AWS Neuron Setup Guide.

Next Step

Using the DLAMI with AWS Neuron

Using the DLAMI with AWS Neuron

A typical workflow with the AWS Neuron SDK is to compile a previously trained machine learning model on a compilation server. After this, distribute the artifacts to the Inf1 instances for execution. AWS Deep Learning AMIs (DLAMI) comes pre-installed with everything you need to compile and run inference in an Inf1 instance that uses Inferentia.

The following sections describe how to use the DLAMI with Inferentia.

Contents

- Using TensorFlow-Neuron and the AWS Neuron Compiler
- Using AWS Neuron TensorFlow Serving
- Using MXNet-Neuron and the AWS Neuron Compiler
- Using MXNet-Neuron Model Serving
- Using PyTorch-Neuron and the AWS Neuron Compiler

Using TensorFlow-Neuron and the AWS Neuron Compiler

This tutorial shows how to use the AWS Neuron compiler to compile the Keras ResNet-50 model and export it as a saved model in SavedModel format. This format is a typical TensorFlow model interchangeable format. You also learn how to run inference on an Inf1 instance with example input.

For more information about the Neuron SDK, see the AWS Neuron SDK documentation.

Contents

- Prerequisites

- Activate the Conda environment
- Resnet50 Compilation
- ResNet50 Inference

Prerequisites

Before using this tutorial, you should have completed the set up steps in Launching a DLAMI Instance with AWS Neuron. You should also have a familiarity with deep learning and using the DLAMI.

Activate the Conda environment

Activate the TensorFlow-Neuron conda environment using the following command:

```
source activate aws_neuron_tensorflow_p36
```

To exit the current conda environment, run the following command:

```
source deactivate
```

Resnet50 Compilation

Create a Python script called **`tensorflow_compile_resnet50.py`** that has the following content. This Python script compiles the Keras ResNet50 model and exports it as a saved model.

```
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')
```

Compile the model using the following command:

```
python tensorflow_compile_resnet50.py
```

The compilation process will take a few minutes. When it completes, your output should look like the following:

```
...
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
...
```

After compilation, the saved model is zipped at **`ws_resnet50/resnet50_neuron.zip`**. Unzip the model and download the sample image for inference using the following commands:

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

ResNet50 Inference

Create a Python script called **`tensorflow_infer_resnet50.py`** that has the following content. This script runs inference on the downloaded model using a previously compiled inference model.

```
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])
```

Run inference on the model using the following command:

```
python tensorflow_infer_resnet50.py
```

Your output should look like the following:

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

Next Step

Using AWS Neuron TensorFlow Serving

Using AWS Neuron TensorFlow Serving

This tutorial shows how to construct a graph and add an AWS Neuron compilation step before exporting the saved model to use with TensorFlow Serving. TensorFlow Serving is a serving system that allows you to scale-up inference across a network. Neuron TensorFlow Serving uses the same API as normal TensorFlow Serving. The only difference is that a saved model must be compiled for AWS Inferentia and the entry point is a different binary named `tensorflow_model_server_neuron`. The binary is found at `/usr/local/bin/tensorflow_model_server_neuron` and is pre-installed in the DLAMI.

For more information about the Neuron SDK, see the AWS Neuron SDK documentation.

Contents

- Prerequisites
- Activate the Conda environment
- Compile and Export the Saved Model
- Serving the Saved Model
- Generate inference requests to the model server

Prerequisites

Before using this tutorial, you should have completed the set up steps in Launching a DLAMI Instance with AWS Neuron. You should also have a familiarity with deep learning and using the DLAMI.

Activate the Conda environment

Activate the TensorFlow-Neuron conda environment using the following command:

```
source activate aws_neuron_tensorflow_p36
```

If you need to exit the current conda environment, run:

```
source deactivate
```

Compile and Export the Saved Model

Create a Python script called `tensorflow-model-server-compile.py` with the following content. This script constructs a graph and compiles it using Neuron. It then exports the compiled graph as a saved model.

```
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)
```

Compile the model using the following command:

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

Your output should look like the following:

```
...
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
```

Serving the Saved Model

Once the model has been compiled, you can use the following command to serve the saved model with the tensorflow_model_server_neuron binary:

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

Your output should look like the following. The compiled model is staged in the Inferentia device’s DRAM by the server to prepare for inference.

```
...
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 ...
```

Generate inference requests to the model server

Create a Python script called `tensorflow-model-server-infer.py` with the following content. This script runs inference via gRPC, which is service framework.

```
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))
```

Run inference on the model by using gRPC with the following command:

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

Your output should look like the following:

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

Using MXNet-Neuron and the AWS Neuron Compiler

The MXNet-Neuron compilation API provides a method to compile a model graph that you can run on an AWS Inferentia device.

In this example, you use the API to compile a ResNet-50 model and use it to run inference.

For more information about the Neuron SDK, see the AWS Neuron SDK documentation.

Contents

- Prerequisites
- Activate the Conda Environment
- Resnet50 Compilation
- ResNet50 Inference

Prerequisites

Before using this tutorial, you should have completed the set up steps in Launching a DLAMI Instance with AWS Neuron. You should also have a familiarity with deep learning and using the DLAMI.

Activate the Conda Environment

Activate the MXNet-Neuron conda environment using the following command:

```
source activate aws_neuron_mxnet_p36
```

To exit the current conda environment, run:

```
source deactivate
```

Resnet50 Compilation

Create a Python script called **`mxnet_compile_resnet50.py`** with the following content. This script uses the MXNet-Neuron compilation Python API to compile a ResNet-50 model.

```
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)
```

Compile the model using the following command:

```
python mxnet_compile_resnet50.py
```

Compilation will take a few minutes. When compilation has finished, the following files will be in your current directory:

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

ResNet50 Inference

Create a Python script called **`mxnet_infer_resnet50.py`** with the following content. This script downloads a sample image and uses it to run inference with the compiled model.

```
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]))
```

Run inference with the compiled model using the following command:

```
python mxnet_infer_resnet50.py
```

Your output should look like the following:

```
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
```

Next Step

[Using MXNet-Neuron Model Serving]

Using MXNet-Neuron Model Serving

In this tutorial, you learn to use a pre-trained MXNet model to perform real-time image classification with Multi Model Server (MMS). MMS is a flexible and easy-to-use tool for serving deep learning models that are trained using any machine learning or deep learning framework. This tutorial includes a compilation step using AWS Neuron and an implementation of MMS using MXNet.

For more information about the Neuron SDK, see the AWS Neuron SDK documentation.

Contents

- Prerequisites
- Activate the Conda Environment
- Download the Example Code
- Compile the Model
- Run Inference

Prerequisites

Before using this tutorial, you should have completed the set up steps in Launching a DLAMI Instance with AWS Neuron. You should also have a familiarity with deep learning and using the DLAMI.

Activate the Conda Environment

Activate the MXNet-Neuron conda environment by using the following command:

```
source activate aws_neuron_mxnet_p36
```

To exit the current conda environment, run:

```
source deactivate
```

Download the Example Code

To run this example, download the example code using the following commands:

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

Compile the Model

Create a Python script called `multi-model-server-compile.py` with the following content. This script compiles the ResNet50 model to the Inferentia device target.

```
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)
```

To compile the model, use the following command:

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

Your output should look like the following:

```
...
[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!
```

Create a file named `signature.json` with the following content to configure the input name and shape:

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

Download the `synset.txt` file by using the following command. This file is a list of names for ImageNet prediction classes.

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

Create a custom service class following the template in the `model_server_template` folder. Copy the template into your current working directory by using the following command:

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

Edit the `mxnet_model_service.py` module to replace the `mx.cpu()` context with the `mx.neuron()` context as follows. You also need to comment out the unnecessary data copy for `model_input` because MXNet-Neuron does not support the NDArray and 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]
```

Package the model with model-archiver using the following commands:

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

Run Inference

Start the Multi Model Server and load the model that uses the RESTful API by using the following commands. Ensure that **neuron-rtd** is running with the default settings.

```
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
```

Run inference using an example image with the following commands:

```
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
```

Your output should look like the following:

```
[
  {
    "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"
  }
]
```

To cleanup after the test, issue a delete command via the RESTful API and stop the model server using the following commands:

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

multi-model-server --stop
```

You should see the following output:

```
{
  "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
```

Using PyTorch-Neuron and the AWS Neuron Compiler

The PyTorch-Neuron compilation API provides a method to compile a model graph that you can run on an AWS Inferentia device.

A trained model must be compiled to an Inferentia target before it can be deployed on Inf1 instances. The following tutorial compiles the torchvision ResNet50 model and exports it as a saved TorchScript module. This model is then used to run inference.

For convenience, this tutorial uses an Inf1 instance for both compilation and inference. In practice, you may compile your model using another instance type, such as the c5 instance family. You must then deploy your compiled model to the Inf1 inference server. For more information, see the AWS Neuron PyTorch SDK Documentation.

Contents

- Prerequisites
- Activate the Conda Environment
- Resnet50 Compilation

- ResNet50 Inference

Prerequisites

Before using this tutorial, you should have completed the set up steps in Launching a DLAMI Instance with AWS Neuron. You should also have a familiarity with deep learning and using the DLAMI.

Activate the Conda Environment

Activate the PyTorch-Neuron conda environment using the following command:

```
source activate aws_neuron_pytorch_p36
```

To exit the current conda environment, run:

```

source deactivate
```

Resnet50 Compilation

Create a Python script called **`pytorch_trace_resnet50.py`** with the following content. This script uses the PyTorch-Neuron compilation Python API to compile a ResNet-50 model.

> **Note**
>
> There is a dependency between versions of torchvision and the torch package that you should be aware of when compiling torchvision models. These dependency rules can be managed through pip. Torchvision==0.6.1 matches the torch==1.5.1 release, while torchvision==0.8.2 matches the torch==1.7.1 release.

```
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")
```

Run the compilation script.

```
python pytorch_trace_resnet50.py
```

Compilation will take a few minutes. When compilation has finished, the compiled model is saved as `resnet50_neuron.pt` in the local directory.

ResNet50 Inference

Create a Python script called **`pytorch_infer_resnet50.py`** with the following content. This script downloads a sample image and uses it to run inference with the compiled model.

```
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) )
```

Run inference with the compiled model using the following command:

```
python pytorch_infer_resnet50.py
```

Your output should look like the following:

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

The ARM64 DLAMI

AWS ARM64 GPU DLAMIs are designed to provide high performance and cost efficiency for deep learning workloads. Specifically, the G5g instance type features the Arm64-based AWS Graviton2 processor, which was built from the ground up by AWS and optimized for how customers run their workloads in the cloud. AWS ARM64 GPU DLAMIs are pre-configured with Docker, NVIDIA Docker, NVIDIA Driver, CUDA, CuDNN, NCCL, as well as popular machine learning frameworks such as TensorFlow and PyTorch.

With the G5g instance type, you can take advantage of the price and performance benefits of Graviton2 to deploy GPU-accelerated deep learning models at a significantly lower cost when compared with x86-based instances with GPU acceleration.

Select a ARM64 DLAMI

Launch a G5g instance with the ARM64 DLAMI of your choice.

For step-by-step instructions on launching a DLAMI, see Launching and Configuring a DLAMI.

For a list of the most recent ARM64 DLAMIs, see the Release Notes for DLAMI.

Get Started

The following topics show you how to get started using the ARM64 DLAMI.

Contents

- Using the ARM64 GPU PyTorch DLAMI

Using the ARM64 GPU PyTorch DLAMI

The AWS Deep Learning AMIs is ready to use with Arm64 processor-based GPUs, and comes optimized for PyTorch. The ARM64 GPU PyTorch DLAMI includes a Python environment pre-configured with PyTorch, TorchVision, and TorchServe for deep learning training and inference use cases.

Contents

- Verify PyTorch Python Environment
- Run Training Sample with PyTorch
- Run Inference Sample with PyTorch

Verify PyTorch Python Environment

Connect to your G5g instance and activate the base Conda environment with the following command:

```
source activate base
```

Your command prompt should indicate that you are working in the base Conda environment, which contains PyTorch, TorchVision, and other libraries.

```
(base) $
```

Verify the default tool paths of the PyTorch environment:

```
(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)
```

Run Training Sample with PyTorch

Run a sample MNIST training job:

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

Your output should look similar to the following:

```
...
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%)
```

Run Inference Sample with PyTorch

Use the following commands to download a pre-trained densenet161 model and run inference using 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
```

Your output should look similar to the following:

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023322898778133094
}
```

Use the following commands to unregister the densenet161 model and stop the server:

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

Your output should look similar to the following:

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

Inference

This section provides tutorials on how to run inference using the DLAMI's frameworks and tools.

Inference Tools

- TensorFlow Serving

Model Serving

The following are model serving options installed on the Deep Learning AMI with Conda. Click on one of the options to learn how to use it.

Topics

- TensorFlow Serving

- TorchServe

TensorFlow Serving

TensorFlow Serving is a flexible, high-performance serving system for machine learning models.

The `tensorflow-serving-api` is pre-installed with Deep Learning AMI with Conda! You will find an example scripts to train, export, and serve an MNIST model in `~/examples/tensorflow-serving/`.

To run any of these examples, first connect to your Deep Learning AMI with Conda and activate the TensorFlow environment.

```
$ source activate tensorflow2_p310
```

Now change directories to the serving example scripts folder.

```
$ cd ~/examples/tensorflow-serving/
```

Serve a Pretrained Inception Model

The following is an example you can try for serving different models like Inception. As a general rule, you need a servable model and client scripts to be already downloaded to your DLAMI.

Serve and Test Inference with an Inception Model

1. Download the model.

   ```
   $ curl -O https://s3-us-west-2.amazonaws.com/tf-test-models/INCEPTION.zip
   ```

2. Untar the model.

   ```
   $ unzip INCEPTION.zip
   ```

3. Download a picture of a husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Launch the server. Note, that for Amazon Linux, you must change the directory used for `model_base_path`, from `/home/ubuntu` to `/home/ec2-user`.

```
$ tensorflow_model_server --model_name=INCEPTION --model_base_path=/home/ubuntu/
examples/tensorflow-serving/INCEPTION/INCEPTION --port=9000
```

5. With the server running in the foreground, you need to launch another terminal session to continue. Open a new terminal and activate TensorFlow with `source activate tensorflow2_p310`. Then use your preferred text editor to create a script that has the following content. Name it `inception_client.py`. This script will take an image filename as a parameter, and get a prediction result from the pre-trained model.

```
from __future__ import print_function

import grpc
import tensorflow as tf
import argparse

from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

parser = argparse.ArgumentParser(
    description='TF Serving Test',
    formatter_class=argparse.ArgumentDefaultsHelpFormatter
)
parser.add_argument('--server_address', default='localhost:9000',
                    help='Tenforflow Model Server Address')
parser.add_argument('--image', default='Siberian_Husky_bi-eyed_Flickr.jpg',
                    help='Path to the image')
args = parser.parse_args()


def main():
  channel = grpc.insecure_channel(args.server_address)
  stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
  # Send request
  with open(args.image, 'rb') as f:
    # See prediction_service.proto for gRPC request/response details.
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'INCEPTION'
    request.model_spec.signature_name = 'predict_images'

    input_name = 'images'
    input_shape = [1]
    input_data = f.read()
    request.inputs[input_name].CopyFrom(
      tf.make_tensor_proto(input_data, shape=input_shape))

    result = stub.Predict(request, 10.0)  # 10 secs timeout
    print(result)

  print("Inception Client Passed")


if __name__ == '__main__':
  main()
```

6. Now run the script passing the server location and port and the husky photo's filename as the parameters.

```
$ python3 inception_client.py --server=localhost:9000 --image Siberian_Husky_bi-
eyed_Flickr.jpg
```

Train and Serve an MNIST Model

For this tutorial we will export a model then serve it with the `tensorflow_model_server` application. Finally, you can test the model server with an example client script.

Run the script that will train and export an MNIST model. As the script's only argument, you need to provide a folder location for it to save the model. For now we can just put it in `mnist_model`. The script will create the folder for you.

```
$ python mnist_saved_model.py /tmp/mnist_model
```

Be patient, as this script may take a while before providing any output. When the training is complete and the model is finally exported you should see the following:

```
Done training!
Exporting trained model to mnist_model/1
Done exporting!
```

Your next step is to run `tensorflow_model_server` to serve the exported model.

```
$ tensorflow_model_server --port=9000 --model_name=mnist --model_base_path=/tmp/
mnist_model
```

A client script is provided for you to test the server.

To test it out, you will need to open a new terminal window.

```
$ python mnist_client.py --num_tests=1000 --server=localhost:9000
```

More Features and Examples

If you are interested in learning more about TensorFlow Serving, check out the TensorFlow website.

You can also use TensorFlow Serving with Amazon Elastic Inference. Check out the guide on how to Use Elastic Inference with TensorFlow Serving for more info.

TorchServe

TorchServe is a flexible tool for serving deep learning models that have been exported from PyTorch. TorchServe comes preinstalled with the Deep Learning AMI with Conda.

For more information on using TorchServe, see Model Server for PyTorch Documentation.

Topics

Serve an Image Classification Model on TorchServe

This tutorial shows how to serve an image classification model with TorchServe. It uses a DenseNet-161 model provided by PyTorch. Once the server is running, it listens for prediction requests. When you upload an image, in this case, an image of a kitten, the server returns a prediction of the top 5 matching classes out of the classes that the model was trained on.

To serve an example image classification model on TorchServe

1. Connect to an Amazon Elastic Compute Cloud (Amazon EC2) instance with Deep Learning AMI with Conda v34 or later.
2. Activate the `pytorch_p310` environment.

   ```
   source activate pytorch_p310
   ```

3. Clone the TorchServe repository, then create a directory to store your models.

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

4. Archive the model using the model archiver. The `extra-files` param uses a file from the `TorchServe` repo, so update the path if necessary. For more information about the model archiver, see Torch Model archiver for 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. Run TorchServe to start an endpoint. Adding `> /dev/null` quiets the log output.

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

6. Download an image of a kitten and send it to the TorchServe predict endpoint:

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

The predict endpoint returns a prediction in JSON similar to the following top five predictions, where the image has a 47% probability of containing an Egyptian cat, followed by a 46% chance it has a tabby cat.

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

7. When you finish testing, stop the server:

```
torchserve --stop
```

Other Examples

TorchServe has a variety of examples that you can run on your DLAMI instance. You can view them on the TorchServe project repository examples page.

More Info

For more TorchServe documentation, including how to set up TorchServe with Docker and the latest TorchServe features, see the TorchServe project pageon GitHub.

Upgrading Your DLAMI

Here you will find information on upgrading your DLAMI and tips on updating software on your DLAMI.

Always keep your operating system and other installed software up to date by applying patches and updates as soon as they become available.

If you are using Amazon Linux or Ubuntu, when you login to your DLAMI, you are notified if updates are available and see instructions for updating. For further information on Amazon Linux maintenance, see Updating Instance Software. For Ubuntu instances, refer to the official Ubuntu documentation.

On Windows, check Windows Update regularly for software and security updates. If you prefer, have updates applied automatically.

Important

For information about the Meltdown and Spectre vulnerabilities and how to patch your operating system to address them, see Security Bulletin AWS-2018-013.

Topics

- Upgrading to a New DLAMI Version
- Tips for Software Updates
- Receive Notifications on New Updates

Upgrading to a New DLAMI Version

DLAMI's system images are updated on a regular basis to take advantage of new deep learning framework releases, CUDA and other software updates, and performance tuning. If you have been using a DLAMI for some time and want to take advantage of an update, you would need to launch a new instance. You would also have to manually transfer any datasets, checkpoints, or other valuable data. Instead, you may use Amazon EBS to retain your data and attach it to a new DLAMI. In this way, you can upgrade often, while minimizing the time it takes to transition your data.

Note

When attaching and moving Amazon EBS volumes between DLAMIs, you must have both the DLAMIs and the new volume in the same Availability Zone.

1. Use the Amazon EC2console to create a new Amazon EBS volume. For detailed directions, see Creating an Amazon EBS Volume.
2. Attach your newly created Amazon EBS volume to your existing DLAMI. For detailed directions, see Attaching an Amazon EBS Volume.
3. Transfer your data, such as datasets, checkpoints, and configuration files.
4. Launch a DLAMI. For detailed directions, see Setting up a DLAMI instance.
5. Detach the Amazon EBS volume from your old DLAMI. For detailed directions, see Detaching an Amazon EBS Volume.
6. Attach the Amazon EBS volume to your new DLAMI. Follow the instructions from the Step 2 to attach the volume.
7. After you verify that your data is available on your new DLAMI, stop and terminate your old DLAMI. For detailed clean-up instructions, see Cleaning up a DLAMI instance.

Tips for Software Updates

From time to time, you may want to manually update software on your DLAMI. It is generally recommended that you use `pip` to update Python packages. You should also use `pip` to update packages within a Conda environment on the Deep Learning AMI with Conda. Refer to the particular framework's or software's website for upgrading and installation instructions.

Note

We cannot guarantee that a package update will be successful. Attempting to update a package in an environment with incompatible dependencies can result in a failure. In such a case, you should contact the library maintainer to see if it is possible to update the package dependencies. Alternatively, you can attempt to modify the environment in such a way that allows the update. However, this modification will likely mean removing or updating existing packages, which means that we can no longer guarantee stability of this environment.

The AWS Deep Learning AMIs comes with many Conda environments and many packages preinstalled. Due to the number of packages preinstalled, finding a set of packages that are guaranteed to be compatible is difficult. You may see a warning "The environment is inconsistent, please check the package plan carefully". DLAMI ensures that all the DLAMI-provided environments are correct, but cannot guarantee that any user installed packages will function correctly.

Receive Notifications on New Updates

Note

AWS Deep Learning AMIs have a weekly release cadence for security patches. Release notifications will be sent for these incremental security patches though they may not be included in official release notes.

You can receive notifications whenever a new DLAMI is released. Notifications are published with Amazon SNS using the following topic.

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

Messages are posted here when a new DLAMI is published. The version, metadata, and regional AMI ID’s of the AMI will be included in the message.

These messages can be received using several different methods. We recommend that you use the following method.

1. Open the Amazon SNS console.
2. In the navigation bar, change the AWS Region to **US West (Oregon)**, if necessary. You must select the region where the SNS notification that you're subscribing to was created.
3. In the navigation pane, choose **Subscriptions, Create subscription**.
4. For the **Create subscription** dialog box, do the following:

    a. For **Topic ARN**, copy and paste the following Amazon Resource Name (ARN): **`arn:aws:sns:us-west-2:767397762724:dlami-updates`**
    b. For **Protocol**, choose one from **[Amazon SQS, AWS Lamda, Email, Email-JSON]**
    c. For **Endpoint**, enter the email address or **Amazon Resource Name (ARN)** of resource that you will use to receive the notifications.

   d. Choose **Create subscription**.

5. You receive a confirmation email with the subject line *AWS Notification - Subscription Confirmation*. Open the email and choose **Confirm subscription** to complete your subscription.

Security in AWS Deep Learning AMIs

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Deep Learning AMIs, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using DLAMI. The following topics show you how to configure DLAMI to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your DLAMI resources.

For more information, see Security in Amazon EC2 in the *Amazon EC2 User Guide*.

Topics

- Data protection in AWS Deep Learning AMIs
- Identity and access management for AWS Deep Learning AMIs
- Compliance validation for AWS Deep Learning AMIs
- Resilience in AWS Deep Learning AMIs
- Infrastructure security in AWS Deep Learning AMIs
- Monitoring AWS Deep Learning AMIs instances

Data protection in AWS Deep Learning AMIs

The AWS shared responsibility model applies to data protection in AWS Deep Learning AMIs. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with DLAMI or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Identity and access management for AWS Deep Learning AMIs

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use DLAMI resources. IAM is an AWS service that you can use with no additional charge.

For more information about identity and access management, see Identity and access management for Amazon EC2.

Topics

- Authenticating with identities
- Managing access using policies
- IAM with Amazon EMR

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access –** To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions –** An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access –** You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access –** Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS) –** When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

IAM with Amazon EMR

You can use IAM with Amazon EMR to define users, AWS resources, groups, roles, and policies. You can also control which AWS services these users and roles can access.

For more information about using IAM with Amazon EMR, see AWS Identity and Access Management for Amazon EMR.

Compliance validation for AWS Deep Learning AMIs

Third-party auditors assess the security and compliance of AWS Deep Learning AMIs as part of multiple AWS compliance programs. For information about the supported compliance programs, see Compliance validation for Amazon EC2.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using DLAMI is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with AWS Config Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices.

Resilience in AWS Deep Learning AMIs

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

For information about Amazon EC2 features to help support your data resiliency and backup needs, see Resilience in Amazon EC2 in the *Amazon EC2 User Guide*.

Infrastructure security in AWS Deep Learning AMIs

The infrastructure security of AWS Deep Learning AMIs is backed by Amazon EC2. For more information, see Infrastructure security in Amazon EC2 in the *Amazon EC2 User Guide*.

Monitoring AWS Deep Learning AMIs instances

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS Deep Learning AMIs instance and your other AWS solutions. Your DLAMI instance comes with several GPU monitoring tools, including a utility that reports GPU usage statistics to Amazon CloudWatch. For more information, see GPU Monitoring and Optimization, and see Monitor Amazon EC2 resources in the *Amazon EC2 User Guide*.

Opting out of usage tracking for DLAMI instances

The following AWS Deep Learning AMIs operating system distributions include code that allows AWS to collect instance type, instance ID, DLAMI type, and OS information.

Note

AWS doesn't collect or retain any other information about the DLAMI, such as the commands that you use within the DLAMI.

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

To opt out of usage tracking

If you choose, you can opt out of usage tracking for a new DLAMI instance. To opt out, you must add a tag to your Amazon EC2 instance during launch. The tag should use the key

`OPT_OUT_TRACKING` with the associated value set to `true`. For more information, see Tag your Amazon EC2 resources in the *Amazon EC2 User Guide*.

DLAMI framework support policy

Here you can find details of the support policy for AWS Deep Learning AMIs (DLAMI) frameworks.

For a list of the DLAMI frameworks that AWS currently supports, see the DLAMI Framework Support Policy page. In the tables on that page, keep in mind the following:

- **Current version** specifies the framework version in the format *x.y.z*. In this format, *x* refers to the major version, *y* refers to the minor version, and *z* refers to the patch version. For example, for TensorFlow 2.10.1, the major version is 2, the minor version is 10, and the patch version is 1.
- **End of patch** specifies how long AWS supports that framework version.

For detailed information about specific DLAMIs, see Release notes for DLAMIs.

DLAMI framework support FAQs

- What framework versions get security patches?
- What images does AWS publish when new framework versions are released?
- What images get new SageMaker AI/AWS features?
- How is current version defined in the Supported Frameworks table?
- What if I am running a version that is not in the Supported Frameworks table?
- Do DLAMIs support previous versions of TensorFlow?
- How can I find the latest patched image for a supported framework version?
- How frequently are new images released?
- Will my instance be patched in place while my workload is running?
- What happens when a new patched or updated framework version is available?
- Are dependencies updated without changing the framework version?
- When does active support for my framework version end?
- Will images with framework versions that are no longer actively maintained be patched?
- How do I use an older framework version?
- How do I stay up-to-date with support changes in frameworks and their versions?
- Do I need a commercial license to use the Anaconda Repository?

What framework versions get security patches?

If the framework version is labeled **Supported** in the AWS Deep Learning AMIs Framework Support Policy table, it gets security patches.

What images does AWS publish when new framework versions are released?

We publish new DLAMIs soon after new versions of TensorFlow and PyTorch are released. This includes major versions, major-minor versions, and major-minor-patch versions of frameworks. We also update images when new versions of drivers and libraries become available. For more information on image maintenance, see When does active support for my framework version end?

What images get new SageMaker AI/AWS features?

New features typically release in the latest version of DLAMIs for PyTorch and TensorFlow. Refer to the release notes for a specific image for details on new SageMaker AI or AWS features. For a list of available DLAMIs, see Release Notes for DLAMI. For more information on image maintenance, see When does active support for my framework version end?

How is current version defined in the Supported Frameworks table?

The current version in the AWS Deep Learning AMIs Framework Support Policy table refers to the newest framework version that AWS makes available on GitHub. Each latest release includes updates to the drivers, libraries, and relevant packages in the DLAMI. For information on image maintenance, see When does active support for my framework version end?

What if I am running a version that is not in the Supported Frameworks table?

If you are running a version that is not in the AWS Deep Learning AMIs Framework Support Policy table, you may not have the most updated drivers, libraries, and relevant packages. For a more up-to-date version, we recommend that you upgrade to one of the supported frameworks available using the latest DLAMI of your choice. For a list of available DLAMIs, see Release Notes for DLAMI.

Do DLAMIs support previous versions of TensorFlow?

No. We support the latest patch version of each framework's latest major version released 365 days from its initial GitHub release as stated in the AWS Deep Learning AMIs Framework Support

Policy table. For more information, see What if I am running a version that is not in the Supported Frameworks table?

How can I find the latest patched image for a supported framework version?

To use a DLAMI with the latest framework version, retrieve the DLAMI ID and use it to launch the DLAMI using the EC2 Console. For sample AWS CLI commands to retrieve the AWS Deep Learning AMIs ID, refer to the DLAMI release notes page single-framework DLAMI release notes. The framework version that you choose must be labeled **Supported** in the AWS Deep Learning AMIs Framework Support Policy table.

How frequently are new images released?

Providing updated patch versions is our highest priority. We routinely create patched images at the earliest opportunity. We monitor for newly patched framework versions (ex. TensorFlow 2.9 to TensorFlow 2.9.1) and new minor release versions (ex. TensorFlow 2.9 to TensorFlow 2.10) and make them available at the earliest opportunity. When an existing version of TensorFlow is released with a new version of CUDA, we release a new DLAMI for that version of TensorFlow with support for the new CUDA version.

Will my instance be patched in place while my workload is running?

No. Patch updates for DLAMI are not “in-place” updates.

You must turn on a new EC2 instance, migrate your workloads and scripts, and then turn off your previous instance.

What happens when a new patched or updated framework version is available?

Regularly check the release notes page for your image. We encourage you to upgrade to new patched or updated frameworks when they are available. For a list of available DLAMIs, see Release Notes for DLAMI.

Are dependencies updated without changing the framework version?

We update dependencies without changing the framework version. However, if a dependency update causes an incompatibility, we create an image with a different version. Be sure to check the Release Notes for DLAMI for updated dependency information.

When does active support for my framework version end?

DLAMI images are immutable. Once they are created they do not change. There are four main reasons why active support for a framework version ends:

- Framework version (patch) upgrades
- AWS security patches
- End of patch date (Aging out)
- Dependency end-of-support

> **Note**
>
> Due to the frequency of version patch upgrades and security patches, we recommend checking the release notes page for your DLAMI often, and upgrading when changes are made.

Framework version (patch) upgrades

If you have a DLAMI workload based on TensorFlow 2.7.0 and TensorFlow releases version 2.7.1 on GitHub, then AWS releases a new DLAMI with TensorFlow 2.7.1. The previous images with 2.7.0 are no longer actively maintained once the new image with TensorFlow 2.7.1 is released. The DLAMI with TensorFlow 2.7.0 does not receive further patches. The DLAMI release notes page for TensorFlow 2.7 is then updated with the latest information. There is no individual release note page for each minor patch.

New DLAMIs created due to patch upgrades are designated with a new AMI ID.

AWS security patches

If you have a workload based on an image with TensorFlow 2.7.0 and AWS makes a security patch, then a new version of the DLAMI is released for TensorFlow 2.7.0. The previous version of the images with TensorFlow 2.7.0 is no longer actively maintained. For more information, see Will my instance be patched in place while my workload is running? For steps on finding the latest DLAMI, see How can I find the latest patched image for a supported framework version?

New DLAMIs created due to patch upgrades are designated with a new AMI ID.

End of patch date (Aging out)

DLAMIs hit their end of patch date 365 days after the GitHub release date.

For multi-framework DLAMIs, when one of the framework versions is updated, a new DLAMI with the updated version is required. The DLAMI with the old framework version is no longer actively maintained.

Important

We make an exception when there is a major framework update. For example. if TensorFlow 1.15 updates to TensorFlow 2.0, then we continue to support the most recent version of TensorFlow 1.15 for a period of two years from the date of the GitHub release or six months after the origin framework maintenance team drops support, whichever date is earlier.

Dependency end-of-support

If you are running a workload on a TensorFlow 2.7.0 DLAMI image with Python 3.6 and that version of Python is marked for end-of-support, then all DLAMI images based on Python 3.6 will no longer be actively maintained. Similarly, if an OS version like Ubuntu 16.04 is marked for end-of-support, then all DLAMI images that are dependent on Ubuntu 16.04 will no longer be actively maintained.

Will images with framework versions that are no longer actively maintained be patched?

No. Images that are no longer actively maintained will not have new releases.

How do I use an older framework version?

To use a DLAMI with an older framework version, retrieve the DLAMI ID and use it to launch the DLAMI using the EC2 Console. For AWS CLI commands to retrieve the AMI ID, refer to the release notes page in the single-framework DLAMI release notes.

How do I stay up-to-date with support changes in frameworks and their versions?

Stay up-to-date with DLAMI frameworks and versions using the AWS Deep Learning AMIs Framework Support Policy table, the DLAMI release notes.

Do I need a commercial license to use the Anaconda Repository?

Anaconda shifted to a commercial licensing model for certain users. Actively maintained DLAMIs have been migrated to the publicly available open-source version of Conda (conda-forge) from the Anaconda channel.

Important NVIDIA driver changes to DLAMIs

On November 15, 2023, AWS made important changes to AWS Deep Learning AMIs (DLAMI) related to the NIVIDA driver that DLAMIs use. For information about what changed and whether it affects your usage of DLAMIs, see DLAMI NVIDIA driver change FAQs.

DLAMI NVIDIA driver change FAQs

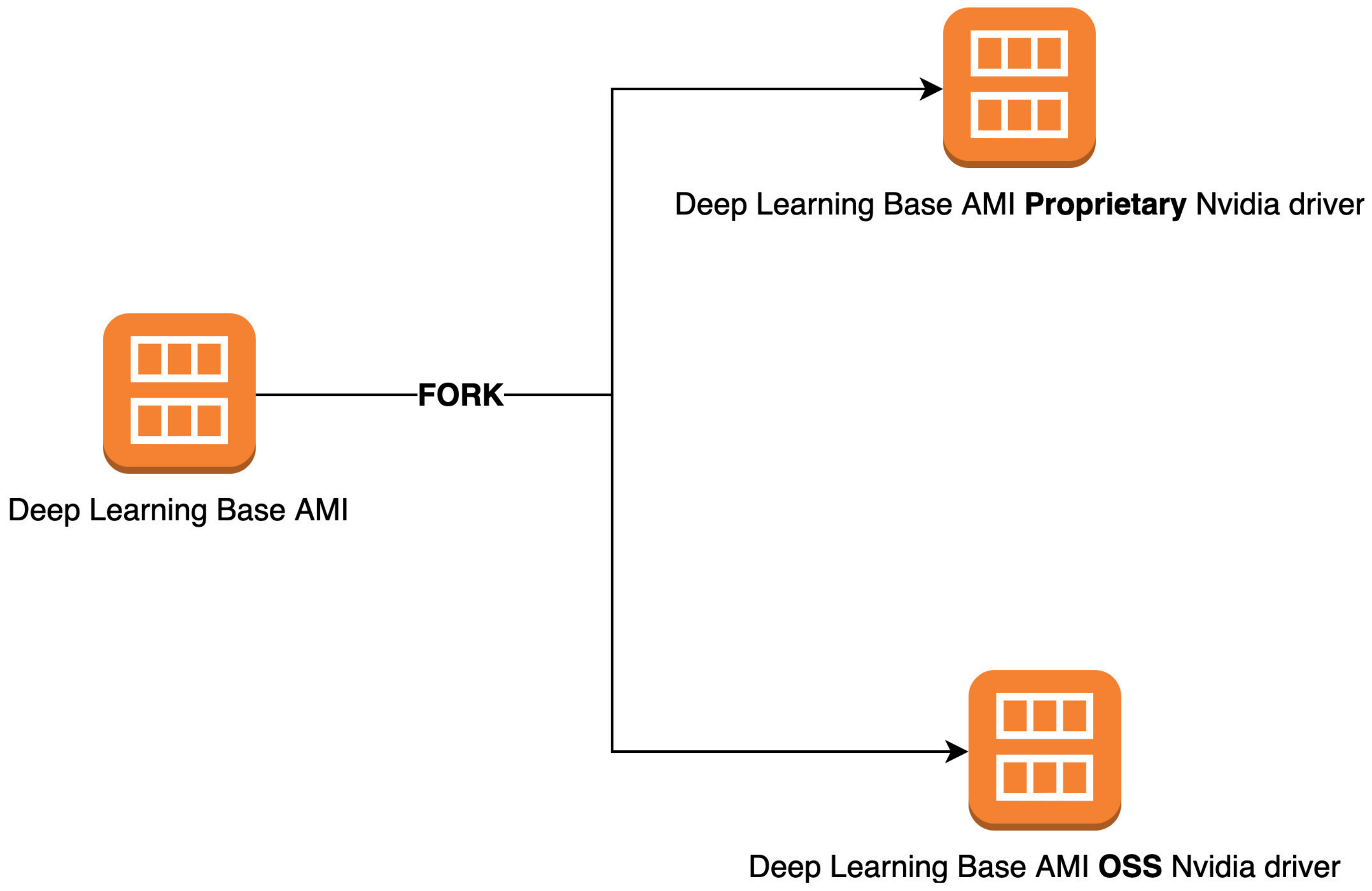
- What changed?
- Why was this change required?
- Which DLAMIs did this change affect?
- What does this mean for you?
- Is there any loss of functionality with the newer DLAMIs?
- Did this change affect Deep Learning Containers?

What changed?

We split DLAMIs into two separate groups:

- DLAMIs that use NVIDIA proprietary driver (to support P3, P3dn, G3)
- DLAMIs that use NVIDIA OSS driver (to support G4dn, G5, P4, P5)

As a result, we created new DLAMIs for each of the two categories with new names and new AMI IDs. These DLAMIs are *not* interchangeable. That is, DLAMIs from one group don't support instances that the other group supports. For example, the DLAMI that supports P5 doesn't support G3, and the DLAMI that supports G3 doesn't support P5.


Deep Learning Base AMI **Proprietary** Nvidia driver
Deep Learning Base AMI
FORK
Deep Learning Base AMI **OSS** Nvidia driver

Why was this change required?

Previously, DLAMIs for NVIDIA GPUs included a proprietary kernel driver from NVIDIA. However, the upstream Linux kernel community accepted a change that isolates proprietary kernel drivers, such as the NVIDIA GPU driver, from communicating with other kernel drivers. This change disables GPUDirect RDMA on P4 and P5 series instances, which is the mechanism that allows GPUs to efficiently use EFA for distributed training. As a result, DLAMIs now use OpenRM driver (NVIDIA open source driver), linked against the open source EFA drivers to support G4dn,G5, P4, and P5. However, this OpenRM driver doesn't support older instances (such as P3 and G3). Therefore, to ensure that we continue to provide current, performant, and secure DLAMIs that support both instance types, we split DLAMIs into two groups: one with the OpenRM driver (that supports G4dn, G5, P4, and P5), and one with the older proprietary driver (that supports P3, P3dn, and G3).

Which DLAMIs did this change affect?

This change affected all DLAMIs.

What does this mean for you?

All DLAMIs will continue to provide functionality, performance, and security as long as you run them on a supported Amazon Elastic Compute Cloud (Amazon EC2) instance type. To determine the EC2 instance types that a DLAMI supports, check the release notes for that DLAMI, and then look for **Supported EC2 Instances**. For a list of currently supported DLAMI options and links to their release notes, see Release notes for DLAMIs.

Additionally, you must use the correct AWS Command Line Interface (AWS CLI) commands to invoke the current DLAMIs.

For base DLAMIs that support P3, P3dn, and G3, use this command:

```
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
```

For base DLAMIs that support G4dn, G5, P4, and P5, use this command:

```
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
```

Is there any loss of functionality with the newer DLAMIs?

No, there is no loss of functionality. The current DLAMIs provide all the functionality, performance, and security of the previous DLAMIs, provided you run them on a supported EC2 instance type.

Did this change affect Deep Learning Containers?

No, this change didn't affect AWS Deep Learning Containers, because they don't include the NVIDIA driver. However, make sure to run Deep Learning Containers on AMIs that are compatible with the underlying instances.

Related information about DLAMI

You can find other resources with related information about DLAMI outside of the AWS Deep Learning AMIs Developer Guide. On AWS re:Post, check out questions about DLAMI from other customers, or ask your own questions. On the AWS Machine Learning Blog and other AWS blogs, read official posts about DLAMI.

AWS re:Post

Tag: AWS Deep Learning AMIs

AWS Blog

- AWS Machine Learning Blog | Category: AWS Deep Learning AMIs
- AWS Machine Learning Blog | Faster training with optimized TensorFlow 1.6 on Amazon EC2 C5 and P3 instances
- AWS Machine Learning Blog | New AWS Deep Learning AMIs for Machine Learning Practitioners
- AWS Partner Network (APN) Blog | New Training Courses Available: Introduction to Machine Learning & Deep Learning on AWS
- AWS News Blog | Journey into Deep Learning with AWS

Deprecated features of DLAMI

The following table lists the deprecated features of AWS Deep Learning AMIs (DLAMI), the date that we deprecated them, and details about why we deprecated them.

Feature	Date	Details
Ubuntu 16.04	10/07/2021	Ubuntu Linux 16.04 LTS reached the end of its five-year LTS window on April 30, 2021 and is no longer supported by its vendor. There are no longer updates to the Deep Learning Base AMI (Ubuntu 16.04) in new releases as of October 2021. Previous releases will continue to be available.
Amazon Linux	10/07/2021	Amazon Linux is end-of-life as of December 2020. There are no longer updates to the Deep Learning AMI (Amazon Linux) in new releases as of October 2021. Previous releases of the Deep Learning AMI (Amazon Linux) will continue to be available.
Chainer	07/01/2020	Chainer has announced the end of major releases as of December, 2019. Consequently, we will no longer include Chainer

Feature	Date	Details
		Conda environments on the DLAMI starting July 2020. Previous releases of the DLAMI that contain these environments will continue to be available . We will provide updates to these environments only if there are security fixes published by the open source community for these frameworks.
Python 3.6	06/15/2020	Due to customer requests, we are moving to Python 3.7 for new TF/MX/PT releases.
Python 2	01/01/2020	The Python open source community has officially ended support for Python 2. The TensorFlow, PyTorch, and MXNet communities have also announced that TensorFlow 1.15, TensorFlo w 2.1, PyTorch 1.4, and MXNet 1.6.0 releases will be the last ones supporting Python 2.

Document history for DLAMI

The following table provides a history of recent DLAMI releases and related changes to the AWS Deep Learning AMIs Developer Guide.

Recent changes

Change	Description	Date
ARM64 DLAMI	The AWS Deep Learning AMIs now supports images on Arm64 processor-based GPUs.	November 29, 2021
TensorFlow 2	The Deep Learning AMI with Conda now comes with TensorFlow 2 with CUDA 10.	December 3, 2019
AWS Inferentia	The Deep Learning AMI now supports AWS Inferenti a hardware and the AWS Neuron SDK.	December 3, 2019
Using TensorFlow Serving with an Inception Model	An example for using inference with an Inception model was added for TensorFlow Serving, for both with and without Elastic Inference.	November 28, 2018
Installing PyTorch from a Nightly Build	A tutorial was added that covers how you can uninstall PyTorch, then install a nightly build of PyTorch on your Deep Learning AMI with Conda.	September 25, 2018
Conda Tutorial	The example MOTD was updated to reflect a more recent release.	July 23, 2018

Earlier changes

The following table provides a history of earlier DLAMI releases and related changes prior to July 2018.

Change	Description	Date
TensorFlow with Horovod	Added a tutorial for training ImageNet with TensorFlow and Horovod.	June 6, 2018
Upgrading guide	Added the upgrading guide.	May 15, 2018
New regions and new 10 minute tutorial	New regions added: US West (N. California), South America, Canada (Central), EU (London), and EU (Paris). Also, the first release of a 10-minute tutorial titled: "Getting Started with Deep Learning AMI".	April 26, 2018
Chainer tutorial	A tutorial for using Chainer in multi-GPU, single GPU, and CPU modes was added. CUDA integration was upgraded from CUDA 8 to CUDA 9 for several frameworks.	February 28, 2018
Linux AMIs v3.0, plus introduction of MXNet Model Server, TensorFlow Serving, and TensorBoard	Added tutorials for Conda AMIs with new model and visualization serving capabilit ies using MXNet Model Server v0.1.5, TensorFlow Serving v1.4.0, and TensorBoard v0.4.0. AMI and framework CUDA capabilities described in Conda and CUDA overviews.	January 25, 2018

Change	Description	Date
	Latest release notes moved to https://aws.amazon.com/re leasenotes/	
Linux AMIs v2.0	Base, Source, and Conda AMIs updated with NCCL 2.1. Source and Conda AMIs updated with MXNet v1.0, PyTorch 0.3.0, and Keras 2.0.9.	December 11, 2017
Two Windows AMI options added	Windows 2012 R2 and 2016 AMIs released: added to AMI selection guide and added to release notes.	November 30, 2017
Initial documentation release	Detailed description of change with link to topic/sec tion that was changed.	November 15, 2017