aws

개발자 안내서

AWS Deep Learning AMIs

AWS Deep Learning AMIs: 개발자 안내서

Table of Contents

AWS Deep Learning AMIs란 무엇인가요?

AWS Deep Learning AMIs (DLAMI)는 클라우드에서 딥 러닝에 사용할 수 있는 사용자 지정 머신 이미지를 제공합니다. DLAMIs는 소형 CPU 전용 인스턴스부터 최신 고성능 다중 GPU 인스턴스에 이르기까지 AWS 리전 다양한 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스 유형에 대해 대부분에서 사용할 수 있습니다. DLAMI는 NVIDIA CUDA 및 NVIDIA cuDNN은 물론 인기 있는 딥 러닝 프레임워크의 최신 릴리스로 사전 구성됩니다.

이 가이드 정보

이 콘텐츠는 DLAMI를 시작하고 사용하는 데 도움이 될 수 있습니다. 이 가이드에서는 훈련과 추론 모두에 대한 몇 가지 일반적인 딥 러닝 사용 사례를 다룹니다. 또한 목적에 맞는 AMI와 선호하는 인스턴스 종류를 선택하는 방법도 다룹니다.

또한 DLAMI에는 지원되는 프레임워크에서 제공하는 여러 자습서가 포함되어 있습니다. 이 가이드는 각 프레임워크를 활성화하는 방법과 적합한 자습서를 찾는 방법을 보여줍니다. 또한 분산 훈련, 디버깅, AWS Inferentia 및 AWS Trainium 사용 및 기타 주요 개념에 대한 자습서도 있습니다. 브라우저에서 자습서를 실행하도록 Jupyter Notebook 서버를 설정하는 방법에 대한 지침은 DLAMI 인스턴스에서 Jupyter Notebook 서버 설정 섹션을 참조하세요.

사전 조건

성공적으로 DLAMI를 실행하기 위해 명령줄 도구와 기본적인 Python을 익히는 것이 좋습니다.

DLAMI 사용 사례 예시

다음은 AWS Deep Learning AMIs (DLAMI)에 대한 몇 가지 일반적인 사용 사례의 예입니다.

딥 러닝 관련 학습 - DLAMI는 기계 학습 및 딥 러닝 프레임워크의 학습 또는 교육을 위한 최적의 선택입니다. DLAMI는 각 프레임워크 설치와 관련된 문제 해결과 동일한 컴퓨터에서 이를 실행하는 방법에 관한 고민을 없애줍니다. DLAMI에는 Jupyter Notebook이 포함되며, 이를 통해 기계 학습과 딥 러닝을 처음 사용하는 사용자를 위해 프레임워크에서 제공되는 자습서를 실행하기가 쉽습니다.

앱 개발 - 딥 러닝 기술로 최신 AI를 활용하는 앱을 개발하고 싶다면 DLAMI를 활용하는 것을 적극 추천합니다. 각 프레임워크에는 딥 러닝을 시작하는 방법에 관한 자습서가 함께 제공되며, 이들 중 다수에는 Model Zoo가 있어 직접 신경망을 생성하거나 모델 교육을 할 필요 없이 딥 러닝을 손쉽게 시도할 수

있습니다. 일부 예제에서는 몇 분 만에 이미지 감지 애플리케이션을 빌드하는 방법 또는 챗봇의 음성 인식 앱을 빌드하는 방법을 보여줍니다.

기계 학습 및 데이터 분석 - 데이터 과학자이거나 딥 러닝을 통해 데이터를 처리하는 데 관심이 있는 경우 R 및 Spark를 지원하는 많은 프레임워크를 확인할 수 있습니다. 개인화 및 예측 시스템에 대한 확장형 데이터 처리 시스템 빌드에 이르는 단순 회귀 작업을 수행하는 방법에 관한 자습서를 찾을 수 있습니다.

연구 - 새 프레임워크를 시도하거나, 새 모델을 테스트하거나, 새 모델을 훈련하려는 연구원이라면 DLAMI와 규모 조정 AWS 기능을 통해 여러 훈련 노드의 지루한 설치 및 관리로 인한 문제를 완화할 수 있습니다.

> **Note**
>
> 처음에는 인스턴스 유형을 GPU가 더 많은(최대 8개) 더 큰 인스턴스로 업그레이드하는 쪽을 선택할 수 있지만, DLAMI 인스턴스 클러스터 생성을 통해 수평적으로 규모를 조정할 수도 있습니다. 클러스터 빌드에 대한 자세한 내용은 DLAMI 관련 정보 섹션을 참조하세요.

DLAMI의 기능

AWS Deep Learning AMIs (DLAMI)의 기능에는 사전 설치된 딥 러닝 프레임워크, GPU 소프트웨어, 모델 서버 및 모델 시각화 도구가 포함됩니다.

사전 설치된 프레임워크

DLAMI 기본 버전은 현재 두 가지이고, 운영 체제(OS) 및 소프트웨어 버전에 따른 기타 변형이 있습니다.

- Conda를 사용하는 Deep Learning AMI - 프레임워크가 설치되어 별도의 conda 패키지 및 별도의 Python 환경 사용
- Deep Learning Base AMI - 프레임워크가 설치되지 않음, NVIDIA CUDA 및 기타 종속성 전용

Conda를 사용하는 Deep Learning AMI는 conda 환경으로 각 프레임워크를 격리합니다. 따라서 원하는 대로 프레임워크 간에 전환할 수 있고, 종속성 충돌을 걱정할 필요가 없습니다. Conda를 사용하는 Deep Learning AMI는 다음 프레임워크를 지원합니다.

- PyTorch

- TensorFlow 2

> **Note**
>
> DLAMI는 더 이상 Apache MXNet, Microsoft Cognitive Toolkit(CNTK), Caffe, Caffe2, Theano, Chainer 및 Keras 딥 러닝 프레임워크를 지원하지 않습니다.

사전 설치된 GPU 소프트웨어

CPU 전용 인스턴스만 사용하더라도 DLAMI에 NVIDIA CUDA 및 NVIDIA cuDNN이 포함됩니다. 설치된 소프트웨어는 인스턴스 유형에 관계없이 동일합니다. GPU 전용 도구는 최소 하나의 GPU가 있는 인스턴스에서만 작동된다는 점에 유의하세요. 인스턴스 유형에 대한 자세한 내용은 DLAMI 인스턴스 유형 선택 섹션을 참조하세요.

CUDA에 대한 자세한 내용은 CUDA 설치 및 프레임워크 바인딩 섹션을 참조하세요.

모델 서비스 및 시각화

Conda를 사용하는 Deep Learning AMI에는 TensorFlow용 모델 서버가 미리 설치되어 있으며, 모델 시각화를 위한 TensorBoard도 함께 제공됩니다. 자세한 내용은 TensorFlow Serving 단원을 참조하십시오.

DLAMI 릴리스 노트

여기서 현재 지원되는 모든 AWS Deep Learning AMIs (DLAMI) 옵션에 대한 자세한 릴리스 정보를 확인할 수 있습니다.

더 이상 지원되지 않는 DLAMI 프레임워크에 대한 릴리스 노트는 DLAMI 프레임워크 지원 정책 페이지의 지원되지 않는 프레임워크 릴리스 노트 아카이브 섹션을 참조하세요.

> **Note**
>
> AWS Deep Learning AMIs 에는 보안 패치에 대한 야간 릴리스 주기가 있습니다. 이러한 증분 보안 패치는 공식 릴리스 노트에는 포함되어 있지 않습니다.

Base DLAMI

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 Deep Learning Base ARM64 AMI(Amazon Linux 2023)

Qualcomm

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

AWS Neuron

- Neuron DLAMI 가이드 참조

단일 프레임워크 DLAMI

PyTorch 전용 AMI

GPU

- X86
  - AWS Deep Learning AMI GPU PyTorch 2.6(Amazon Linux 2023)
  - AWS Deep Learning AMI GPU PyTorch 2.6(Ubuntu 22.04)
  - AWS Deep Learning AMI GPU PyTorch 2.5(Amazon Linux 2023)
  - AWS Deep Learning AMI GPU PyTorch 2.5(Ubuntu 22.04)
  - 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)
- ARM64
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.6(Amazon Linux 2023)
  - AWS Deep Learning ARM64 AMI GPU PyTorch 2.6(Ubuntu 22.04)
  - 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 Neuron

- Neuron DLAMI 가이드 참조

TensorFlow 전용 AMI

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 Neuron

- Neuron DLAMI 가이드 참조

다중 프레임워크 DLAMI

> **Tip**
> 기계 학습 프레임워크를 하나만 사용하는 경우 단일 프레임워크 DLAMI를 사용하는 것이 좋습니다.

GPU

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

AWS Neuron

- Neuron DLAMI 가이드 참조

DLAMI 시작하기

이 설명서에는 적절한 DLAMI 선택, 사용 사례 및 예산에 맞는 인스턴스 유형 선택에 대한 팁과 유용한 DLAMI 관련 정보 사용자 지정 설정을 설명합니다.

Amazon EC2를 처음 AWS 사용하거나 사용하는 경우 로 시작합니다Conda를 사용하는 Deep Learning AMI. Amazon EC2 및 Amazon EMR, Amazon EFS 또는 Amazon S3와 같은 기타 AWS 서비스에 익숙하고 분산 훈련 또는 추론이 필요한 프로젝트에 대해 이러한 서비스를 통합하는 데 관심이 있는 경우 사용 사례에 DLAMI 관련 정보 적합한지 확인하세요.

DLAMI 선택 단원을 확인하여 어떤 인스턴스 유형이 애플리케이션에 가장 적절한지에 관한 아이디어를 얻는 것이 좋습니다.

다음 단계

DLAMI 선택

DLAMI 선택

GPU DLAMI 릴리스 정보에 언급된 대로 다양한 DLAMI 옵션을 제공합니다. 사용 목적에 맞는 올바른 DLAMI를 선택할 수 있도록 개발된 하드웨어 유형 또는 기능별로 이미지를 그룹화합니다. 최상위 그룹 분류는 다음과 같습니다.

- DLAMI 유형: 기본, 단일 프레임워크, 다중 프레임워크(Conda DLAMI)
- 컴퓨팅 아키텍처: x86 기반, Arm64-based AWS Graviton
- 프로세서 유형: GPU, CPU, Inferentia, Trainium
- SDK: CUDA, AWS Neuron
- OS: Amazon Linux, Ubuntu

이 설명서의 나머지 섹션에서 유용한 세부 정보를 확인하세요.

주제

- CUDA 설치 및 프레임워크 바인딩
- Deep Learning Base AMI
- Conda를 사용하는 Deep Learning AMI

- DLAMI 아키텍처 옵션
- DLAMI 운영 체제 옵션

다음

Conda를 사용하는 Deep Learning AMI

CUDA 설치 및 프레임워크 바인딩

딥 러닝은 매우 최첨단이긴 하지만 각 프레임워크는 "안정적인" 버전을 제공합니다. 안정적인 버전은 최신 CUDA 또는 cuDNN 구현 및 기능을 사용할 수 없을 수 있습니다. 사용 사례와 필요한 기능은 프레임워크를 선택하는 데 도움이 될 수 있습니다. 잘 모르겠으면 Conda를 사용하는 최신 Deep Learning AMI를 사용하세요. 각 프레임워크에서 지원되는 최신 버전을 사용하며 CUDA를 사용하는 모든 프레임워크의 공식 `pip` 바이너리가 있습니다. 최신 버전을 사용하여 딥 러닝 환경을 사용자 지정하려는 경우 Deep Learning Base AMI를 사용하세요.

자세한 내용은 안정적 후보와 릴리스 후보 비교의 지침을 참조하십시오.

CUDA를 사용하는 DLAMI 선택

Deep Learning Base AMI에는 사용 가능한 CUDA 버전 시리즈가 모두 있습니다.

Conda를 사용하는 Deep Learning AMI에는 사용 가능한 CUDA 버전 시리즈가 모두 있습니다.

> **Note**
>
> MXNet, CNTK, Caffe, Caffe2, Theano, Chainer 또는 Keras Conda 환경은 더 이상 AWS Deep Learning AMIs에 포함되지 않습니다.

특정 프레임워크 버전 번호는 DLAMI 릴리스 노트 섹션을 참조하세요.

이 DLAMI 유형을 선택하거나 다음 옵션에서 다른 DLAMI에 대해 자세히 알아보세요.

CUDA 버전 중 하나를 선택하고 부록에서 DLAMI 전체 목록을 확인하거나 다음 옵션에서 다른 DLAMI에 대해 자세히 알아보세요.

다음

Deep Learning Base AMI

관련 항목

- CUDA 버전 간 전환에 대한 지침은 Deep Learning Base AMI 사용 자습서를 참조하세요.

Deep Learning Base AMI

Deep Learning Base AMI는 딥 러닝을 위한 빈 캔버스와 같습니다. 특정 프레임워크의 설치 시점까지 필요한 모든 것이 제공되며 사용자가 선택한 CUDA 버전이 제공됩니다.

Base DLAMI를 선택해야 하는 이유

이 AMI 그룹은 딥 러닝 프로젝트 사용과 최신 빌드를 원하는 프로젝트 기고자에게 유용합니다. 어느 프레임워크 및 버전을 설치하고자 하는지에 집중할 수 있도록 최신 NVIDIA 소프트웨어가 설치되고 실행 중인 고유한 환경을 안심하고 작동시키고자 하는 경우 유용합니다.

이 DLAMI 유형을 선택하거나 다음 옵션으로 다른 DLAMI에 대해 자세히 알아보세요.

다음

Conda를 사용하는 DLAMI

관련 항목

- Deep Learning Base AMI 사용

Conda를 사용하는 Deep Learning AMI

Conda DLAMI는 conda 가상 환경을 사용하며 다중 프레임워크 또는 단일 프레임워크 DLAMI가 있습니다. 이러한 환경을 다른 프레임워크 설치를 별도로 유지하고 프레임워크간 전환을 간소화하도록 구성됩니다. DLAMI이 제공해야 하는 모든 프레임워크를 통한 학습 및 실험에 매우 유용합니다. 대부분의 사용자가 Conda를 사용하는 새로운 Deep Learning AMI를 최적으로 생각합니다.

프레임워크로부터 최신 버전이 자주 업데이트되고, 최신 GPU 드라이버 및 소프트웨어를 보유합니다. 일반적으로 AWS Deep Learning AMIs 대부분의 문서에서 라고 합니다. 이러한 DLAMIs Ubuntu 20.04, Ubuntu 22.04, Amazon Linux 2, Amazon Linux 2023 운영 체제를 지원합니다. 운영 체제 지원은 업스트림 OS의 지원에 따라 달라집니다.

안정적 후보와 릴리스 후보 비교

Conda AMI는 각 프레임워크의 최신 공식 릴리스의 최적화된 바이너리를 사용합니다. 릴리스 후보 기능과 시험 사용 기능은 발표 예정이 없습니다. 최적화는 인텔의 MKL DNN과 같은 가속화 기술에 대한 프레임워크의 지원에 따라 다릅니다. 이러한 기술은 C5 및 C4 CPU 인스턴스 유형에서 교육 및 추론 속도를 높입니다. 바이너리 또한 컴파일되어 고급 인텔 명령 세트를 지원합니다. 여기에는 AVX, AVX-2, SSE4.1 및 SSE4.2가 포함되지만 이에 국한되지는 않습니다. 이는 인텔 CPU 아키텍처의 가속 벡터 및 부동 소수점 작업입니다. 추가로 GPU 인스턴스 유형의 경우 CUDA 및 cuDNN이 최신 공식 릴리스 지원으로 업데이트됩니다.

Conda를 사용하는 Deep Learning AMI는 프레임워크의 최초 정품 인증 시 EC2 Amazon 인스턴스에 대한 프레임워크의 가장 최적화된 버전을 설치합니다. 자세한 정보는 Conda를 사용하는 Deep Learning AMI 사용 섹션을 참조하세요.

사용자 지정 옵션이나 최적화된 빌드 옵션으로 소스에서 직접 설치하려는 경우에는 Deep Learning Base AMI가 더 나은 옵션이 될 수도 있습니다.

Python 2 사용 중단

Python 오픈 소스 커뮤니티는 2020년 1월 1일 Python 2에 대한 지원을 공식적으로 종료했습니다. TensorFlow 및 PyTorch 커뮤니티는 TensorFlow 2.1 및 PyTorch 1.4 릴리스가 Python 2를 지원하는 마지막 릴리스라고 공지했습니다. Python 2 Conda 환경을 포함하는 DLAMI(v26, v25 등)의 이전 릴리스는 계속 사용할 수 있습니다. 그러나 이전에 게시된 DLAMI 버전의 Python 2 Conda 환경 업데이트는 해당 버전에 대한 오픈 소스 커뮤니티에서 게시한 보안 수정 사항이 있는 경우에만 제공합니다. TensorFlow 및 PyTorch 프레임워크의 최신 버전이 포함된 DLAMI 릴리스는 Python 2 Conda 환경을 포함하지 않습니다.

CUDA 지원

구체적인 CUDA 버전 번호는 GPU DLAMI 릴리스 노트에서 확인할 수 있습니다.

다음

DLAMI 아키텍처 옵션

관련 항목

- Conda를 사용하는 Deep Learning AMI를 사용하는 방법은 Conda를 사용하는 Deep Learning AMI 사용 자습서를 참조하세요.

DLAMI 아키텍처 옵션

AWS Deep Learning AMIs는 x86 기반 또는 Arm64 기반 AWS Graviton2 아키텍처와 함께 제공됩니다.

ARM64 GPU DLAMI 시작에 대한 자세한 내용은 ARM64 DLAMI을 참조하세요. 사용 가능한 인스턴스 유형에 대한 자세한 내용은 DLAMI 인스턴스 유형 선택을 참조하세요.

다음

DLAMI 운영 체제 옵션

DLAMI 운영 체제 옵션

DLAMI는 다음 운영 체제에서 제공됩니다.

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

이전 버전의 운영 체제는 더 이상 사용되지 않는 DLAMI에서 사용할 수 있습니다. DLAMI 사용 중단에 대한 자세한 내용은 DLAMI 사용 중단을 참조하세요.

DLAMI를 선택하기 전에 필요한 인스턴스 유형을 평가하고 AWS 리전을 지정하세요.

다음

DLAMI 인스턴스 유형 선택

DLAMI 인스턴스 유형 선택

보다 일반적으로, DLAMI의 인스턴스 유형을 선택할 때는 다음 사항을 고려해야 합니다.

- 딥 러닝에 익숙하지 않다면 단일 GPU가 있는 인스턴스가 더 적합할 수 있습니다.
- 예산이 한정되어 있다면 CPU 전용 인스턴스를 사용합니다.
- 딥 러닝 모델 추론을 위해 고성능 및 비용 효율성을 최적화하려는 경우 AWS Inferentia 칩과 함께 인스턴스를 사용할 수 있습니다.

- Arm64 기반 CPU 아키텍처를 갖춘 고성능 GPU 인스턴스를 찾고 있다면 G5g 인스턴스 유형을 사용합니다.
- 추론 및 예측을 위해 사전 학습된 모델을 실행하려는 경우 Amazon Elastic Inference를 Amazon EC2 인스턴스에 연결합니다. Amazon Elastic Inference를 사용하면 GPU보다 훨씬 적은 비용으로 가속기에 액세스할 수 있습니다.
- 대용량 추론 서비스의 경우 메모리가 많은 단일 CPU 인스턴스 또는 이러한 인스턴스로 구성된 클러스터가 더 나은 솔루션일 수 있습니다.
- 데이터가 많거나 배치 크기가 큰 대형 모델을 사용하는 경우 더 많은 메모리를 가진 더 큰 인스턴스가 필요합니다. 또한 모델을 GPU의 클러스터에 배포할 수도 있습니다. 배치 크기를 줄이는 경우 적은 메모리를 가진 인스턴스를 사용하는 것이 더 나은 방법일 수 있습니다. 이는 정확도 및 훈련 속도에 영향을 줄 수 있습니다.
- 규모에 맞는 높은 수준의 노드 간 통신이 필요한 NCCL(NVIDIA Collective Communications Library)를 사용하여 기계 학습 애플리케이션을 실행하려면 Elastic Fabric Adapter(EFA)를 사용할 수도 있습니다.

인스턴스에 대한 자세한 내용은 EC2 인스턴스 유형을 참조하세요.

다음 주제에서는 인스턴스 유형 고려 사항에 대한 정보를 제공합니다.

> **Important**
>
> Deep Learning AMI에는 NVIDIA Corporation에서 개발, 소유 또는 제공하는 드라이버, 소프트웨어 또는 도구 키트가 포함되어 있습니다. NVIDIA 드라이버, 소프트웨어 또는 도구 키트를 NVIDIA 하드웨어가 포함된 Amazon EC2 인스턴스에서만 사용하기로 동의합니다.

주제

DLAMI 요금

DLAMI에 포함된 딥 러닝 프레임워크는 무료이며, 각 프레임워크에는 고유한 오픈 소스 라이선스가 있습니다. DLAMI에 포함된 소프트웨어는 무료이지만 기본 Amazon EC2 인스턴스 하드웨어에 대한 비용은 지불해야 합니다.

일부 Amazon EC2 인스턴스 유형은 무료로 제공됩니다. 이러한 무료 인스턴스 중 하나에서 DLAMI를 실행할 수 있습니다. 즉, 인스턴스의 용량만을 사용하는 경우 DLAMI는 완전히 무상으로 제공됩니다. 더 많은 CPU 코어, 디스크 공간, RAM 또는 하나 이상의 GPU를 포함하여 더욱 강력한 인스턴스를 원하는 경우 대부분의 인스턴스가 프리 티어 인스턴스 클래스에 있지 않을 것입니다.

인스턴스 옵션 및 요금에 대한 자세한 내용은 Amazon EC2 요금을 참조하세요.

DLAMI 리전 가용성

각 리전은 다양한 범위의 인스턴스를 지원하며 인스턴스 유형별 비용은 은 리전마다 달라질 수 있습니다. DLAMI는 모든 리전에서 사용 가능한 것은 아니지만 원하는 리전에 DLAMI를 복사하는 것은 가능합니다. 자세한 내용은 AMI 복사를 참조하세요. 리전 선택 목록을 참고하고 사용자와 고객에게 가까운 리전을 선택하세요. 하나 이상의 DLAMI를 사용하고 잠재적으로 클러스터를 생성할 계획이 있는 경우 클러스터의 모든 노드에 대해 동일한 리전을 사용해야 합니다.

리전에 대한 자세한 내용은 Amazon EC2 서비스 엔드포인트를 참조하세요.

다음

권장 GPU 인스턴스

권장 GPU 인스턴스

대부분의 딥 러닝 목적에 GPU 인스턴스가 부합합니다. CPU 인스턴스에 비해 GPU 인스턴스에서의 새로운 모델 교육이 더 빠릅니다. 여러 GPU 인스턴스가 있는 경우 또는 GPU를 포함한 여러 인스턴스에 걸쳐 분산 교육을 사용하는 경우 부선형적으로 확장할 수 있습니다.

DLAMI를 지원하는 인스턴스 유형은 아래를 참조하세요. GPU 인스턴스 유형 옵션 및 사용에 대한 자세한 내용은 EC2 인스턴스 유형에서 가속 컴퓨팅을 참조하세요.

> **Note**
>
> 모델의 크기를 고려하여 인스턴스를 선택해야 합니다. 모델이 인스턴스의 사용 가능한 RAM을 초과하는 경우 애플리케이션을 위해 충분한 메모리를 가진 다른 인스턴스 유형을 선택합니다.

- Amazon EC2 P5e 인스턴스에는 최대 8개의 NVIDIA Tesla H200 GPU가 있습니다.
- Amazon EC2 P5 인스턴스에는 최대 8개의 NVIDIA Tesla H100 GPU가 있습니다.
- Amazon EC2 P4 인스턴스에는 최대 8개의 NVIDIA Tesla A100 GPU가 있습니다.
- Amazon EC2 P3 인스턴스에는 최대 8개의 NVIDIA Tesla V100 GPU가 있습니다.
- Amazon EC2 G3 인스턴스에는 최대 4개의 NVIDIA Tesla M60 GPU가 있습니다.
- Amazon EC2 G4 인스턴스에는 최대 4개의 NVIDIA T4 GPU가 있습니다.
- Amazon EC2 G5 인스턴스에는 최대 8개의 NVIDIA A10G GPU가 있습니다.
- Amazon EC2 G6 인스턴스에는 최대 8개의 NVIDIA L4 GPU가 있습니다.
- Amazon EC2 G6e 인스턴스에는 최대 8개의 NVIDIA L40S Tensor Core GPU가 있습니다.
- Amazon EC2 G5g 인스턴스에는 Arm64 기반 AWS Graviton2 프로세서가 탑재되어 있습니다.

DLAMI 인스턴스는 GPU 프로세스를 모니터링하고 최적화하는 도구를 제공합니다. GPU 프로세스 모니터링에 대한 자세한 내용은 GPU 모니터링 및 최적화를 참조하세요.

G5g 인스턴스 사용에 대한 자세한 내용은 ARM64 DLAMI 자습서를 참조하세요.

다음

권장 CPU 인스턴스

권장 CPU 인스턴스

예산이 한정적이고, 딥 러닝을 학습 중이거나 예측 서비스를 실행하고자 하는 경우 CPU 범주에 저렴한 옵션이 있습니다. 일부 프레임워크는 Intel의 MKL DNN을 활용합니다. 이는 C5(모든 리전에서 사용 가능하지 않음) CPU 인스턴스 유형의 훈련 및 추론 속도를 높입니다. CPU 인스턴스 유형에 대한 자세한 내용은 EC2 인스턴스 유형에서 컴퓨팅 최적화를 참조하세요.

> **Note**
>
> 모델의 크기를 고려하여 인스턴스를 선택해야 합니다. 모델이 인스턴스의 사용 가능한 RAM을 초과하는 경우 애플리케이션을 위해 충분한 메모리를 가진 다른 인스턴스 유형을 선택합니다.

- Amazon EC2 C5 인스턴스에는 최대 72개의 Intel vCPU가 있습니다. C5 인스턴스는 과학 모델링, 배치 처리, 분산 분석, 고성능 컴퓨팅(HPC), 기계학습 및 딥 러닝 추론에 있어 뛰어납니다.

다음

권장 Inferentia 인스턴스

권장 Inferentia 인스턴스

AWS Inferentia 인스턴스는 딥 러닝 모델 추론 워크로드에 높은 성능과 비용 효율성을 제공하도록 설계되었습니다. 특히 Inf2 인스턴스 유형은 TensorFlow 및 PyTorch와 같은 인기 있는 기계 학습 프레임워크와 통합된 AWS Inferentia 칩 및 AWS Neuron SDK를 사용합니다.

고객은 Inf2 인스턴스를 사용하여 검색, 추천 엔진, 컴퓨터 비전, 음성 인식, 자연어 처리, 개인화, 사기 탐지와 같은 대규모 기계 학습 추론 애플리케이션을 클라우드에서 최저 비용으로 실행할 수 있습니다.

> **Note**
>
> 모델의 크기를 고려하여 인스턴스를 선택해야 합니다. 모델이 인스턴스의 사용 가능한 RAM을 초과하는 경우 애플리케이션을 위해 충분한 메모리를 가진 다른 인스턴스 유형을 선택합니다.

- Amazon EC2 Inf2 인스턴스에는 최대 16개의 AWS Inferentia 칩과 100Gbps의 네트워킹 처리량이 있습니다.

AWS Inferentia DLAMIsDLAMI를 사용하는 AWS Inferentia 칩.

다음

권장 Trainium 인스턴스

권장 Trainium 인스턴스

AWS Trainium 인스턴스는 딥 러닝 모델 추론 워크로드에 높은 성능과 비용 효율성을 제공하도록 설계되었습니다. 특히 Trn1 인스턴스 유형은 TensorFlow 및 PyTorch와 같은 인기 있는 기계 학습 프레임워크와 통합된 AWS Trainium 칩 및 AWS Neuron SDK를 사용합니다.

고객은 Trn1 인스턴스를 사용하여 검색, 추천 엔진, 컴퓨터 비전, 음성 인식, 자연어 처리, 개인화, 사기 탐지와 같은 대규모 기계 학습 추론 애플리케이션을 클라우드에서 최저 비용으로 실행할 수 있습니다.

> **Note**
>
> 모델의 크기를 고려하여 인스턴스를 선택해야 합니다. 모델이 인스턴스의 사용 가능한 RAM을 초과하는 경우 애플리케이션을 위해 충분한 메모리를 가진 다른 인스턴스 유형을 선택합니다.

- Amazon EC2 Trn1 인스턴스에는 최대 16개의 AWS Trainium 칩과 100Gbps의 네트워킹 처리량이 있습니다.

DLAMI 인스턴스 설정

사용하려는 DLAMI를 선택하고 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스 유형을 선택하면 새 DLAMI 인스턴스를 설정할 준비가 됩니다.

DLAMI 및 EC2 인스턴스 유형을 아직 선택하지 않은 경우 DLAMI 시작하기 섹션을 참조하세요.

주제

- DLAMI의 ID 찾기
- DLAMI 인스턴스 시작
- DLAMI 인스턴스에 연결
- DLAMI 인스턴스에서 Jupyter Notebook 서버 설정
- DLAMI 인스턴스 정리

DLAMI의 ID 찾기

각 DLAMI에는 고유 식별자(ID)가 있습니다. Amazon EC2 콘솔을 사용하여 DLAMI 인스턴스를 시작할 때 선택적으로 DLAMI ID를 사용하여 사용하려는 DLAMI를 검색할 수 있습니다. AWS Command Line Interface (AWS CLI)를 사용하여 DLAMI 인스턴스를 시작할 때이 ID가 필요합니다.

Amazon EC2 또는의 기능인 Parameter Store에 대한 AWS CLI 명령을 사용하여 선택한 DLAMI의 ID를 찾을 수 있습니다 AWS Systems Manager. 설치 및 구성에 대한 지침은 AWS Command Line Interface 사용 설명서의 시작하기 AWS CLI를 AWS CLI참조하세요.

Using Parameter Store

ssm get-parameter를 사용하여 DLAMI를 찾으려면

다음 ssm get-parameter 명령에서 `--name` 옵션의 파라미터 이름 형식은 *`/aws/service/deeplearning/ami/$Architecture/$ami_type/latest/ami-id`* 입니다. 이 이름 형식에서 *`architecture`*는 **`x86_64`** 또는 **`arm64`**일 수 있습니다. DLAMI 이름을 사용하고 'deep', 'learning' 및 'ami' 키워드를 제거하여 *`ami_type`*을 지정합니다. AMI 이름은 DLAMI 릴리스 노트에서 찾을 수 있습니다.

> **Important**
>
> 이 명령을 사용하려면 사용하는 AWS Identity and Access Management (IAM) 보안 주체에 `ssm:GetParameter` 권한이 있어야 합니다. IAM 위탁자에 대한 자세한 내용은 IAM 사용 설명서에서 IAM 역할의 추가 리소스 섹션을 참조하세요.

- ```
  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
  ```

  다음과 같이 출력됩니다

  ```
  ami-09ee1a996ac214ce7
  ```

  > **Tip**
  >
  > 현재 지원되는 일부 DLAMI 프레임워크의 경우 더 구체적인 예제 ssm get-parameter 명령을 DLAMI 릴리스 노트에서 찾을 수 있습니다. 선택한 DLAMI의 릴리스 노트 링크를 선택한 다음 릴리스 노트에서 해당 ID 쿼리를 찾습니다.

Using Amazon EC2 CLI

ec2 describe-images를 사용하여 DLAMI를 찾으려면

다음 ec2 describe-images 명령에서 Name=name 필터의 값에 DLAMI 이름을 입력합니다. 지정된 프레임워크의 릴리스 버전을 지정하거나 버전 번호를 물음표(?)로 대체하여 최신 릴리스를 가져올 수 있습니다.

- ```
  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
  ```

  다음과 같이 출력됩니다

```
ami-09ee1a996ac214ce7
```

> **Tip**
>
> 선택한 DLAMI와 관련된 예제 ec2 describe-images 명령은 DLAMI 릴리스 노트 섹션을 참조하세요. 선택한 DLAMI의 릴리스 노트 링크를 선택한 다음 릴리스 노트에서 해당 ID 쿼리를 찾습니다.

다음 단계

DLAMI 인스턴스 시작

DLAMI 인스턴스 시작

DLAMI 인스턴스를 시작하는 데 사용할 DLAMI의 ID를 찾으면 인스턴스를 시작할 준비가 된 것입니다. Amazon EC2 콘솔 또는 AWS Command Line Interface ()를 사용하여 시작할 수 있습니다AWS CLI.

> **Note**
>
> 이 연습에서는 Deep Learning Base OSS Nvidia Driver GPU AMI(Ubuntu 22.04)와 관련된 참조를 만듭니다. 다른 DLAMI를 선택한 경우에도 이 설명서를 따를 수 있습니다.

EC2 console

> **Note**
>
> 고성능 컴퓨팅(HPC) 및 기계 학습 애플리케이션을 가속화하기 위해 Elastic Fabric Adapter(EFA)를 사용하여 DLAMI 인스턴스를 시작할 수 있습니다. 관련 지침은 EFA를 사용하여 AWS Deep Learning AMIs 인스턴스 시작 섹션을 참조하세요.

1. EC2 콘솔을 엽니다.
2. 최상위 탐색 AWS 리전 창에 현재를 기록해 둡니다. 원하는 리전이 아닌 경우 진행 전에 이 옵션을 변경하세요. 자세한 내용은 Amazon Web Services 일반 참조의 Amazon EC2 서비스 엔드포인트를 참조하세요.

3. 인스턴스 시작을 선택합니다.
4. 인스턴스 이름을 입력하고 적합한 DLAMI를 선택합니다.

   a. 내 AMI에서 기존 DLAMI를 찾거나 빠른 시작을 선택합니다.

   b. DLAMI ID로 검색합니다. 옵션을 찾은 후 선택합니다.
5. 인스턴스 유형을 선택합니다. DLAMI 릴리스 노트에서 DLAMI에 권장되는 인스턴스 패밀리를 찾을 수 있습니다. DLAMI 인스턴스 유형에 대한 일반적인 권장 사항은 DLAMI 인스턴스 유형 선택 섹션을 참조하세요.
6. 인스턴스 시작을 선택합니다.

AWS CLI

- 를 사용하려면 사용하려는 DLAMI의 ID, AWS 리전 및 EC2 인스턴스 유형, 보안 토큰 정보가 AWS CLI있어야 합니다. 그런 다음 ec2 run-instances AWS CLI 명령을 사용하여 인스턴스를 시작할 수 있습니다.

  설치 및 구성에 대한 지침은 AWS Command Line Interface 사용 설명서의 시작하기 AWS CLI를 AWS CLI참조하세요. 명령 예제를 비롯한 자세한 내용은 AWS CLI를 사용한 Amazon EC2 인스턴스 시작, 나열 및 닫기를 참조하세요.

Amazon EC2 콘솔 또는를 사용하여 인스턴스를 시작한 후 인스턴스가 준비될 때까지 AWS CLI기다립니다. 이는 보통 몇 분 정도 소요됩니다. Amazon EC2 콘솔에서 인스턴스의 상태를 확인할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 상태 확인을 참조하세요.

다음 단계

DLAMI 인스턴스에 연결

DLAMI 인스턴스에 연결

DLAMI 인스턴스를 시작하고 인스턴스가 실행 중이면 SSH를 사용하여 클라이언트(Windows, macOS 또는 Linux)에서 인스턴스에 연결할 수 있습니다. 관련 지침은 Amazon EC2 사용 설명서의 SSH를 사용하여 Linux 인스턴스에 연결을 참조하세요.

로그인한 후 Jupyter Notebook 서버를 설정하는 경우를 위해 SSH 로그인 명령의 사본을 보관하세요. Jupyter 웹페이지에 연결하려면 해당 명령의 변형을 사용합니다.

다음 단계

DLAMI 인스턴스에서 Jupyter Notebook 서버 설정

DLAMI 인스턴스에서 Jupyter Notebook 서버 설정

Jupyter Notebook 서버를 사용하면 DLAMI 인스턴스에서 Jupyter Notebook을 만들고 실행할 수 있습니다. Jupyter 노트북을 사용하면 AWS 인프라를 사용하고 DLAMI에 내장된 패키지에 액세스하는 동안 훈련 및 추론을 위한 기계 학습(ML) 실험을 수행할 수 있습니다. Jupyter Notebook에 대한 자세한 내용은 Jupyter 사용자 설명서 웹사이트에 있는 Jupyter Notebook을 참조하세요.

Jupyter Notebook 서버를 설정하려면 다음을 수행해야 합니다.

- DLAMI 인스턴스에서 Jupyter Notebook 서버를 구성합니다.
- Jupyter Notebook 서버에 연결하도록 클라이언트를 구성합니다. Windows, macOS 및 Linux 클라이언트에 대한 구성 지침이 제공됩니다.
- Jupyter Notebook 서버에 로그인하여 설정을 테스트합니다.

이러한 단계를 완료하려면 다음 항목의 지침을 따르세요. Jupyter Notebook 서버를 설정한 후 DLAMI에 포함된 예제 노트북 자습서를 실행할 수 있습니다. 자세한 내용은 Jupyter Notebook 자습서 실행 단원을 참조하십시오.

주제

- DLAMI 인스턴스에서 Jupyter Notebook 서버 보호
- DLAMI 인스턴스에서 Jupyter Notebook 서버 시작
- DLAMI 인스턴스의 Jupyter Notebook 서버에 클라이언트 연결
- DLAMI 인스턴스에서 Jupyter Notebook 서버에 로그인

DLAMI 인스턴스에서 Jupyter Notebook 서버 보호

Jupyter Notebook 서버를 안전하게 유지하려면 암호를 설정하고 서버에 대한 SSL 인증서를 생성하는 것이 좋습니다. 암호 및 SSL을 구성하려면 먼저 DLAMI 인스턴스에 연결한 후에 다음 지침을 따릅니다.

Jupyter Notebook 서버를 보호하려면

1. Jupyter는 암호 유틸리티를 제공합니다. 다음 명령을 실행하고 프롬프트에 원하는 암호를 입력하십시오.

   ```
   $ jupyter notebook password
   ```

   결과는 다음과 비슷합니다.

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

2. 자체 서명된 SSL 인증서를 생성합니다. 지시에 따라 귀하의 지역을 기입하십시오. 프롬프트를 비워 두려면 .를 입력해야 합니다. 귀하의 답변은 인증서의 기능에 영향을 미치지 않습니다.

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

> **Note**
>
> 서드 파티에서 서명되고 브라우저에서 보안 경고를 표시하지 않는 일반 SSL 인증서를 만들기를 원할 수 있습니다. 이 과정은 훨씬 더 복잡합니다. 자세한 내용은 Jupyter Notebook 사용자 설명서의 노트북 서버 보호를 참조하세요.

다음 단계

DLAMI 인스턴스에서 Jupyter Notebook 서버 시작

DLAMI 인스턴스에서 Jupyter Notebook 서버 시작

암호 및 SSL로 Jupyter Notebook 서버를 보호한 후 서버를 시작할 수 있습니다. DLAMI 인스턴스에 로그인하고 이전에 생성한 SSL 인증서를 사용하는 다음 명령을 실행합니다.

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

서버를 시작한 상태에서 클라이언트 컴퓨터의 SSH 터널을 통해 연결할 수 있습니다. 서버가 실행되면 Jupyter에서 서버가 실행 중임을 확인하는 출력값 일부가 보입니다. 이 시점에서 로컬 호스트 URL을 통해 서버에 액세스할 수 있다는 콜아웃을 무시합니다. 터널을 만들기 전에는 작동하지 않기 때문입니다.

> **Note**
>
> Jupyter 웹 인터페이스를 사용하여 프레임워크를 전환할 때 Jupyter가 환경 변경을 처리합니다. 자세한 내용은 Jupyter를 사용하여 환경 전환 단원을 참조하십시오.

다음 단계

DLAMI 인스턴스의 Jupyter Notebook 서버에 클라이언트 연결

DLAMI 인스턴스의 Jupyter Notebook 서버에 클라이언트 연결

DLAMI 인스턴스에서 Jupyter Notebook 서버를 시작한 후 서버에 연결하도록 Windows, macOS 또는 Linux 클라이언트를 구성합니다. 연결할 때 작업 영역의 서버에서 Jupyter Notebook을 생성 및 액세스하고, 서버에서 딥 러닝 코드를 실행할 수 있습니다.

사전 조건

SSH 터널 설정에 필요한 다음 정보가 있어야 합니다.

- Amazon EC2 인스턴스의 퍼블릭 DNS 이름 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 호스트 이름 유형을 참조하세요.
- 프라이빗 키 파일에 대한 키 페어 키 페어 액세스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 키 페어 및 Amazon EC2 인스턴스를 참조하세요.

Windows, macOS 또는 Linux 클라이언트에서 연결

Windows, macOS 또는 Linux 클라이언트에서 DLAMI 인스턴스에 연결하려면 해당 클라이언트 운영 체제에 대한 지침을 따르세요.

Windows

SSH를 사용하여 Windows 클라이언트에서 DLAMI 인스턴스에 연결하려면

1. PuTTY와 같은 Windows용 SSH 클라이언트를 사용합니다. 관련 지침은 Amazon EC2 사용 설명서의 PuTTY를 사용하여 Linux 인스턴스에 연결을 참조하세요. 다른 SSH 연결 옵션은 SSH를 사용하여 Linux 인스턴스에 연결을 참조하세요.
2. (선택 사항) 실행 중인 Jupyter 서버에 대한 SSH 터널을 생성합니다. Windows 클라이언트에 Git Bash를 설치한 다음 macOS 및 Linux 클라이언트에 대한 연결 지침을 따릅니다.

macOS or Linux

SSH를 사용하여 macOS 또는 Linux 클라이언트에서 DLAMI 인스턴스에 연결하려면

1. 터미널을 엽니다.
2. 다음 명령을 실행하여 로컬 포트 8888의 모든 요청을 원격 Amazon EC2 인스턴스의 포트 8888로 전달합니다. Amazon EC2 인스턴스에 액세스하는 키의 위치와 Amazon EC2 인스턴스의 공개 DNS 이름을 바꾸어 명령을 업데이트합니다. 참고: Amazon Linux AMI의 경우 사용자 이름은 `ubuntu`가 아니라 `ec2-user`입니다.

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

이 명령은 클라이언트와 Jupyter Notebook 서버를 실행 중인 원격 Amazon EC2 인스턴스 사이의 터널을 엽니다.

다음 단계

DLAMI 인스턴스에서 Jupyter Notebook 서버에 로그인

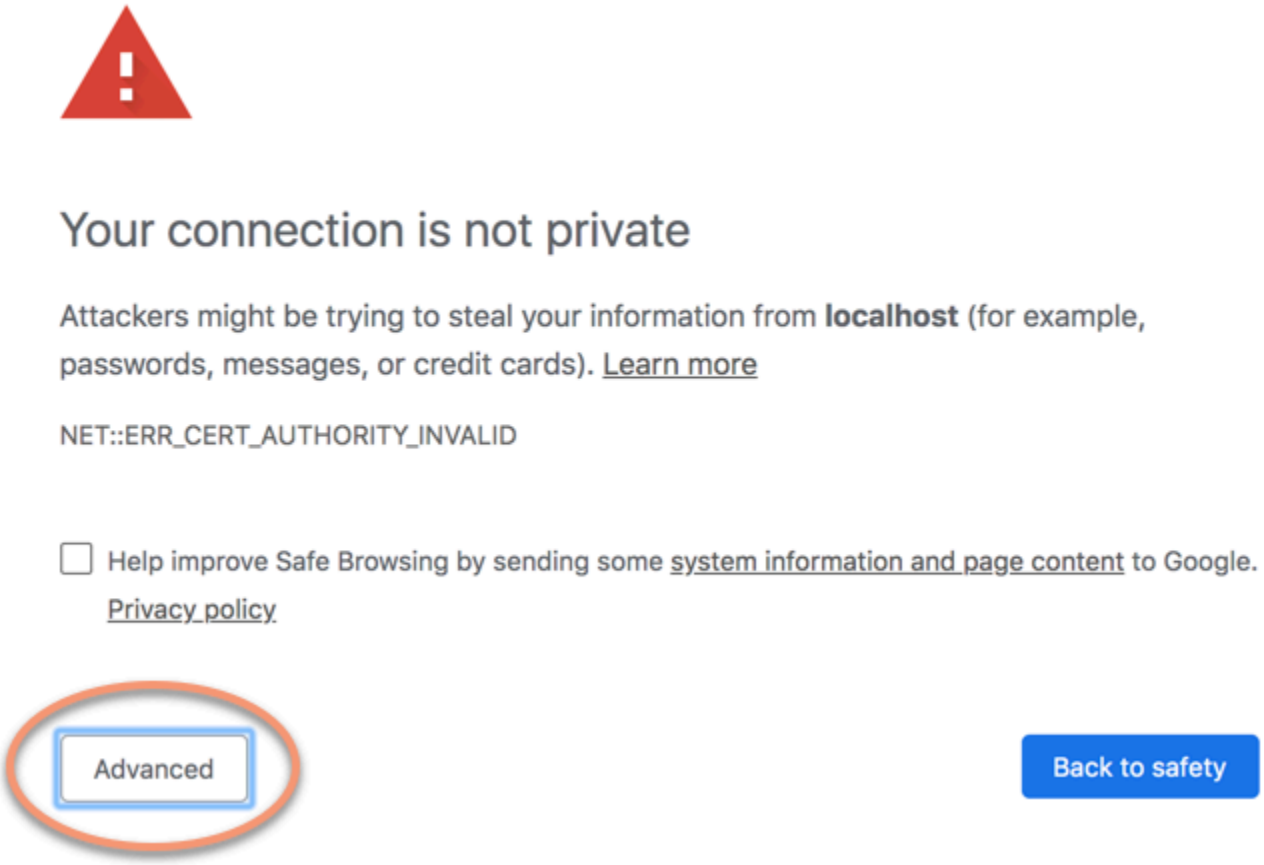
DLAMI 인스턴스에서 Jupyter Notebook 서버에 로그인

클라이언트를 DLAMI 인스턴스의 Jupyter Notebook 서버에 연결한 후 서버에 로그인할 수 있습니다.

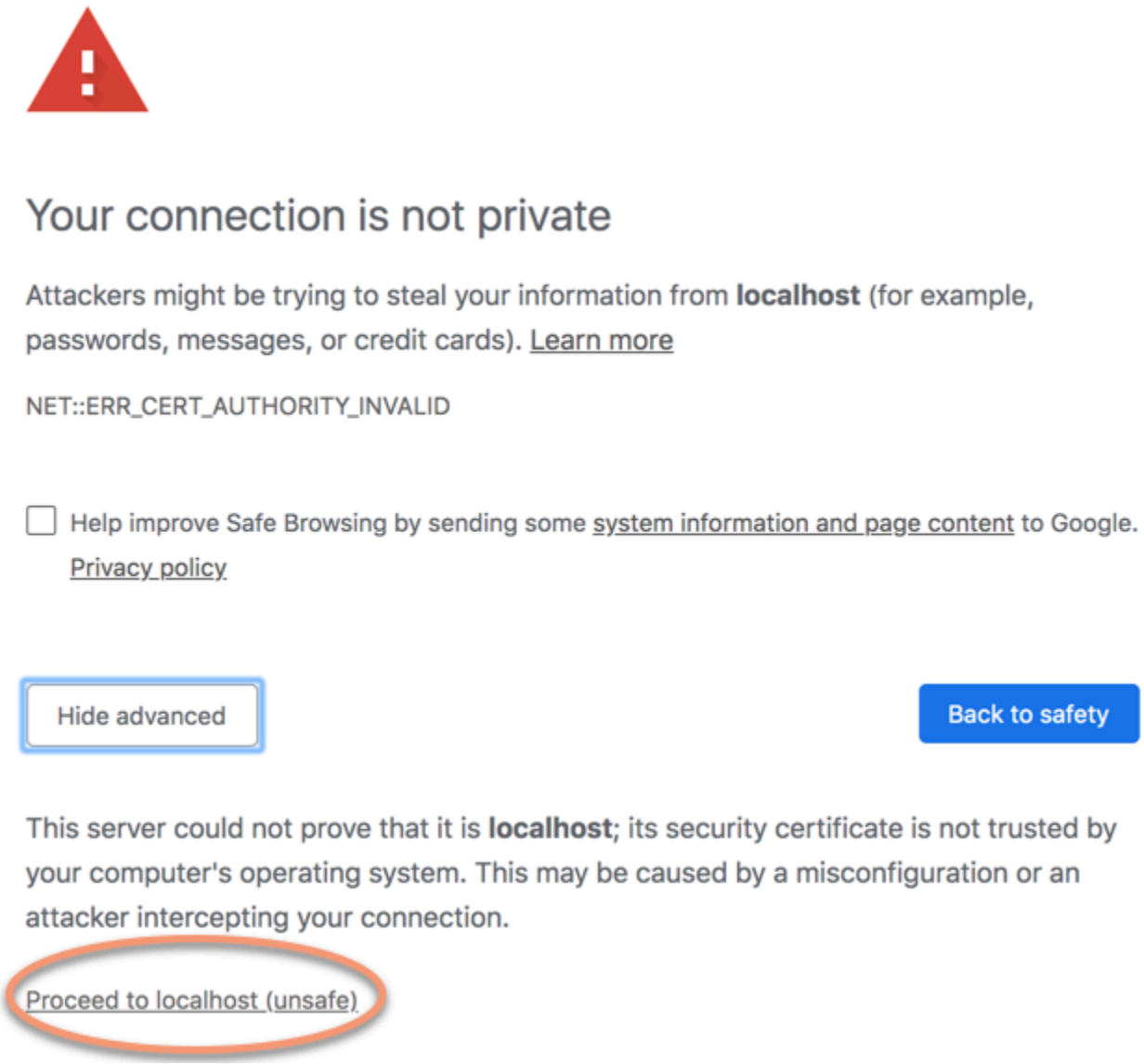
브라우저에서 서버에 로그인하려면

1. 브라우저의 주소 막대에서 https://localhost:8888 URL을 입력하거나 이 링크를 클릭합니다.

2. 자체 서명된 SSL 인증서를 사용하면 브라우저에서 경고 메시지를 표시하고 웹 사이트를 계속 방문하지 않도록 안내합니다.



이것을 스스로 설정했으므로 계속해도 안전합니다. 브라우저에 따라 “고급”, “세부 정보 표시” 또는 유사한 버튼이 표시됩니다.



이것을 클릭한 다음 "localhost로 진행" 링크를 클릭하십시오. 연결이 성공적인 경우 Jupyter Notebook 서버 웹페이지가 보입니다. 이 시점에서 이전에 설정한 암호를 묻는 메시지가 나타납니다.

이제 DLAMI 인스턴스에서 실행 중인 Jupyter Notebook 서버에 액세스할 수 있습니다. 새 노트북을 생성하거나 자습서에서 제공된 노트북을 실행할 수 있습니다.

DLAMI 인스턴스 정리

DLAMI 인스턴스가 더 이상 필요하지 않을 때 Amazon EC2에서 이를 중지하거나 종료하여 예기치 않은 요금이 발생하지 않도록 할 수 있습니다.

인스턴스를 중지하는 경우 인스턴스를 계속 유지했다가 나중에 다시 사용하고 싶을 때 시작할 수 있습니다. 구성, 파일 및 기타 비휘발성 정보가 Amazon Simple Storage Service(Amazon S3)의 볼륨에 저장됩니다. 인스턴스가 중지된 동안에는 볼륨 유지에 대한 S3 요금이 발생하지만 컴퓨팅 리소스에 대한 요금은 발생하지 않습니다. 인스턴스를 다시 시작하면 데이터와 함께 해당 스토리지 볼륨이 탑재됩니다.

인스턴스를 종료하는 경우 인스턴스가 사라지고 다시 시작할 수 없습니다. 물론 종료된 인스턴스의 컴퓨팅 리소스에 대한 요금은 더 이상 발생하지 않습니다. 그러나 데이터는 여전히 Amazon S3에 상주하므로 S3 요금이 계속 발생할 수 있습니다. 종료된 인스턴스와 관련된 추가 요금을 모두 방지하려면 Amazon S3에서 스토리지 볼륨도 삭제해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 종료를 참조하세요.

`stopped` 및 `terminated`와 같은 Amazon EC2 인스턴스 상태에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 상태 변경을 참조하세요.

DLAMI(Deep Learning AMI) 사용

주제

다음 섹션에서는 Conda를 사용하는 DLAMI를 사용하여 환경을 전환하고, 각 프레임워크의 샘플 코드를 실행하고, Jupyter를 실행하여 다양한 노트북 자습서를 시험할 수 있는 방법을 설명합니다.

Conda를 사용하는 Deep Learning AMI 사용

주제

Conda를 사용하는 Deep Learning AMI 소개

Conda는 Windows, macOS 및 Linux에서 실행되는 오픈 소스 패키지 관리 시스템 및 환경 관리 시스템입니다. Conda는 패키지 및 그 종속성을 빠르게 설치, 실행 및 업데이트합니다. Conda는 손쉬운 생성, 저장, 로드 및 로컬 컴퓨터 환경 사이의 전환이 가능합니다.

Conda를 사용하는 Deep Learning AMI는 딥 러닝 환경을 손쉽게 전환할 수 있도록 구성되었습니다. 다음 지침은 conda를 사용한 몇 가지 기본명령을 안내합니다. 프레임워크의 기본 가져오기가 작동 중인지 여부와 프레임워크를 통해 여러 단순 작업을 실행할 수 있는지 여부를 확인하는 데 도움이 됩니다. 그리고 DLAMI와 함께 제공되는 더 많은 자습서 또는 각 프레임워크의 프로젝트 사이트에 있는 프레임워크 예제로 넘어갈 수 있습니다.

DLAMI 로그인

서버에 로그인한 다음 각기 다른 딥 러닝 프레임워크 사이를 전환할 수 있는 다양한 conda 명령을 설명하는 서버 MOTD(Message of the Day)가 보입니다. 아래는 예제 MOTD입니다. 특정 MOTD는 릴리스된 DLAMI 새 버전에 따라 달라질 수 있습니다.

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

TensorFlow 환경 시작

> **Note**
>
> Conda 환경을 처음 시작할 때 로드되는 동안 기다려야 합니다. Conda를 사용하는 Deep Learning AMI는 프레임워크의 최초 정품 인증 시 EC2 인스턴스에 대한 프레임워크의 가장 최적화된 버전을 설치합니다. 이후에는 지연되지 않습니다.

1. Python 3에 대한 TensorFlow 가상 환경을 활성화합니다.

```
$ source activate tensorflow2_p310
```

2. iPython 터미널을 시작합니다.

```
(tensorflow2_p310)$ ipython
```

3. 빠른 TensorFlow 프로그램을 실행합니다.

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

"Hello, Tensorflow!"가 표시됩니다.

다음

Jupyter Notebook 자습서 실행

PyTorch Python 3 환경으로 전환

계속해서 iPython 콘솔에서 `quit()`을 사용하는 경우, 환경을 전환할 준비를 합니다.

- Python 3에 대한 PyTorch 가상 환경을 활성화합니다.

```
$ source activate pytorch_p310
```

일부 PyTorch 코드 테스트

설치를 테스트하려면 Python을 사용하여 어레이를 생성 및 출력하는 PyTorch 코드를 작성합니다.

1. iPython 터미널을 시작합니다.

```
(pytorch_p310)$ ipython
```

2. PyTorch 가져오기

```
import torch
```

타사 패키지에 관한 경고 메시지가 표시될 수 있습니다. 이 서명은 무시할 수 있습니다.

3. 무작위로 초기화된 요소를 포함하는 5x3 매트릭스를 생성합니다. 어레이를 출력합니다.

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

결과를 확인합니다.

```
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]])
```

환경 제거

DLAMI의 공간이 부족하면 사용하고 있지 않은 Conda 패키지를 제거하는 방법을 선택할 수도 있습니다.

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

Deep Learning Base AMI 사용

Deep Learning Base AMI 사용

Base AMI는 GPU 드라이버 및 액셀러레이션 라이브러리의 기본 플랫폼과 함께 제공되어 사용자 지정 딥 러닝 환경을 배포합니다. 기본적으로 AMI는 한 가지 NVIDIA CUDA 버전 환경으로 구성됩니다. 다른 버전의 CUDA 간에 전환할 수도 있습니다. 실행 방법은 다음 지침을 참조하세요.

CUDA 버전 구성

NVIDIA의 nvcc 프로그램을 실행하여 CUDA 버전을 확인할 수 있습니다.

```
nvcc --version
```

다음과 같은 bash 명령을 사용하여 특정 CUDA 버전을 선택하고 확인할 수 있습니다.

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

자세한 내용은 Base DLAMI 릴리스 정보를 참조하세요.

Jupyter Notebook 자습서 실행

각 딥 러닝 프로젝트의 소스에 자습서와 예제가 함께 제공되며, 대부분의 경우 모든 DLAMI 에 적용됩니다. Conda를 사용하는 Deep Learning AMI을(를) 선택했다면 설정 및 시험 사용이 준비된 일부 자습서로부터 추가 혜택을 받게 됩니다.

> **Important**
>
> DLAMI에 설치된 Jupyter Notebook 자습서를 실행하려면 DLAMI 인스턴스에서 Jupyter Notebook 서버 설정 단계를 먼저 완료하세요.

Jupyter 서버가 실행 중이면 웹 브라우저를 통해 자습서를 실행할 수 있습니다. Conda를 사용하는 DLAMI를 실행 중이거나 Python 환경을 설정한 경우 Jupyter Notebook 인터페이스에서 Python 커널을 전환할 수 있습니다. 프레임워크별 자습서를 따라하기 전에 적절한 커널을 선택하세요. 이에 대한 추가 예제는 Conda를 사용하는 DLAMI 사용자에게 제공됩니다.

> **Note**
>
> 많은 자습서의 경우 추가적인 Python 모듈이 필요하지만 DLAMI에 설정되지 않았을 수 있습니다. `"xyz module not found"`와 같은 오류가 발생하는 경우 DLAMI에 로그인하여 위에 설명한 환경을 활성화한 다음 필요한 모듈을 설치하세요.

> **Tip**
>
> 딥 러닝 자습서 및 예제는 주로 하나 이상의 GPU를 필요로 합니다. 인스턴스 유형에 GPU가 없는 경우 일부 예제의 코드를 변경하여 이를 실행해야 할 수 있습니다.

설치된 자습서 탐색

Jupyter 서버에 로그인하면 자습서 디렉터리(Conda를 사용하는 DLAMI 한정)를 찾을 수 있습니다. 각 프레임워크 이름에 따라 자습서 폴더가 표시됩니다. 프레임워크 목록이 표시되지 않은 경우 현재 DLAMI의 프레임워크에서 자습서를 사용할 수 없습니다. 프레임워크의 이름을 클릭하면 자습서가 나열되고, 자습서를 클릭하면 이를 시작할 수 있습니다.

Conda를 사용하는 DLAMI에서 처음 노트북을 실행할 때 어떤 환경을 사용하고자 하는지 물을 것입니다. 선택할 목록이 표시됩니다. 각 환경은 이 패턴에 따라 이름이 지정됩니다.

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

예를 들어 `Environment (conda_mxnet_p36)`을 볼 수 있는데, 이는 환경에 MXNet 및 Python 3가 있음을 간주합니다. 이에 대한 기타 변형은 `Environment (conda_mxnet_p27)`이며, 이는 환경에 MXNet 및 Python 2가 있음을 간주합니다.

> **Tip**
>
> 어떤 버전의 CUDA가 활성 상태인지 고려하는 경우 이를 확인하는 방법은 처음 DLAMI에 로그인할 때의 MOTD에 있습니다.

Jupyter를 사용하여 환경 전환

다른 프레임워크에 대한 자습서를 시험하기로 한 경우 현재 실행 중인 커널을 확인해야 합니다. 이 정보는 Jupyter 인터페이스의 오른쪽 상단, 로그아웃 버튼 아래에서 확인할 수 있습니다. Jupyter 메뉴 항목인 [Kernel]과 [Change Kernel]을 클릭한 다음 실행 중인 노트북에 맞는 환경을 클릭함으로써 열린 노트북의 커널을 변경할 수 있습니다.

이 시점에서 커널의 변경 사항이 이전에 실행했던 것의 상태를 삭제하기 때문에 셀을 재실행해야 합니다.

> **Tip**
>
> 프레임워크 사이의 전환은 재미있고 교육적일 수 있지만 메모리가 부족할 수도 있습니다. 오류가 발생하기 시작하는 경우 Jupyter 서버를 실행 중인 터미널 창을 확인하세요. 유용한 메시지와 오류 로깅이 여기에 표시되며, 메모리 부족 오류를 확인할 수 있습니다. 이를 수정하려면 Jupyter 서버의 홈 페이지로 이동하여 [Running] 탭을 클릭하고 여전히 백그라운드에서 실행

중이어서 모든 메모리를 잡아먹을 수 있는 각 자습서에 대해 [Shutdown]을 클릭할 수 있습니다.

자습서

다음은 Conda를 사용하는 DLAMI 소프트웨어 사용 방법에 대한 자습서입니다.

주제

- 프레임워크 활성화
- Elastic Fabric Adapter를 사용한 분산 훈련
- GPU 모니터링 및 최적화
- DLAMI를 사용하는 AWS Inferentia 칩
- ARM64 DLAMI
- Inference
- 모델 제공

프레임워크 활성화

다음은 Conda를 사용하는 DLAMI에 설치된 딥 러닝 프레임워크입니다. 프레임워크를 클릭하면 활성화하는 방법을 자세히 알아볼 수 있습니다.

주제

- PyTorch
- TensorFlow 2

PyTorch

PyTorch 활성화

프레임워크의 안정적인 Conda 패키지가 출시되면 테스트 후 DLAMI에 사전 설치됩니다. 테스트되지 않은 최신 야간 구축을 실행하려는 경우 수동으로 PyTorch의 야간 구축 설치(실험)를 수행할 수 있습니다.

현재 설치된 프레임워크를 활성화하려면 Conda를 사용하는 DLAMI에서 다음 지침을 따릅니다.

CUDA 및 MKL-DNN이 있는 Python 3의 PyTorch인 경우 다음 명령을 실행하세요.

```
$ source activate pytorch_p310
```

iPython 터미널을 시작합니다.

```
(pytorch_p310)$ ipython
```

빠른 PyTorch 프로그램을 실행합니다.

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

출력된 최초 임의 어레이와 그 키기, 그리고 또 다른 임의 어레이가 추가된 것을 볼 수 있습니다.

PyTorch의 야간 구축 설치(실험)

야간 구축에서 PyTorch를 설치하는 방법

Conda를 사용하는 DLAMI의 두 가지 PyTorch Conda 환경 중 하나 또는 모두에 최신 PyTorch 빌드를 설치할 수 있습니다.

1. • (Python 3에 대한 옵션) - Python 3 PyTorch 환경을 활성화합니다.

   ```
   $ source activate pytorch_p310
   ```

2. 나머지 단계에서는 `pytorch_p310` 환경을 사용하고 있다고 가정합니다. 현재 설치된 PyTorch를 제거합니다.

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

3. • (GPU 인스턴스에 대한 옵션) - CUDA.0이 포함된 PyTorch의 최신 야간 빌드를 설치합니다.

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

   • (CPU 인스턴스에 대한 옵션) - GPU가 없는 인스턴스에 대한 PyTorch의 최신 야간 구축을 설치합니다.

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

4. 최신 야간 구축을 성공적으로 설치했는지 확인하려면 IPython 터미널을 시작하고 PyTorch의 버전을 점검합니다.

```
(pytorch_p310)$ ipython
```

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

출력은 `1.0.0.dev20180922`와 비슷하게 인쇄됩니다.

5. PyTorch 야간 구축이 MNIST 예제에 효과적으로 작동하는지 확인하려면 PyTorch의 예제 리포지토리에서 테스트 스크립트를 실행할 수 있습니다.

```
(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
```

추가 자습서

추가 자습서와 예제는 프레임워크의 공식 설명서, PyTorch 설명서, 그리고 PyTorch 웹 사이트를 참조하세요.

TensorFlow 2

이 자습서는 Conda를 사용하는 DLAMI를 실행 중인 인스턴스에서 TensorFlow 2를 활성화하고 TensorFlow 2 프로그램을 실행하는 방법을 보여줍니다.

프레임워크의 안정적인 Conda 패키지가 출시되면 테스트 후 DLAMI에 사전 설치됩니다.

TensorFlow 2 활성화

Conda를 사용하는 DLAMI 에서 TensorFlow 실행

1. TensorFlow 2를 활성화하려면 Conda를 사용하는 DLAMI의 Amazon Elastic Compute Cloud (Amazon EC2) 인스턴스를 엽니다.

2. CUDA 10.1 및 MKL-DNN이 있는 Python 3의 TensorFlow 2 및 Keras 2의 경우 다음 명령을 실행하세요.

```
$ source activate tensorflow2_p310
```

3. iPython 터미널을 시작합니다:

```
(tensorflow2_p310)$ ipython
```

4. TensorFlow 2 프로그램을 실행하여 제대로 작동하는지 확인합니다.

```


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

화면에 `Hello, TensorFlow!`가 표시되어야 합니다.

추가 자습서

더 많은 자습서와 예제는 TensorFlow Python API에 대한 TensorFlow 설명서 또는 TensorFlow 웹 사이트를 참조하세요.

Elastic Fabric Adapter를 사용한 분산 훈련

Elastic Fabric Adapter(EFA)는 DLAMI 인스턴스에 연결하여 고성능 컴퓨팅(HPC) 및 기계 학습 애플리케이션의 속도를 높일 수 있는 네트워크 디바이스입니다. EFA를 사용하면 AWS 클라우드에서 제공하는 확장성, 유연성 및 탄력성을 통해 온프레미스 HPC 클러스터의 애플리케이션 성능을 달성할 수 있습니다.

다음 섹션에서는 DLAMI에서 EFA 사용을 시작하는 방법을 보여 줍니다.

> **Note**
>
> 이 기본 GPU DLAMI 목록에서 원하는 DLAMI를 선택하세요.

주제

- EFA를 사용하여 AWS Deep Learning AMIs 인스턴스 시작
- DLAMI에서 EFA 사용

EFA를 사용하여 AWS Deep Learning AMIs 인스턴스 시작

최신 버전 Base DLAMI는 EFA와 사용 가능하며, 필수 드라이버, 커널 모듈, libfabric, openmpi 및 GPU 인스턴스용 NCCL OFI 플러그인이 함께 제공됩니다.

지원되는 Base DLAMI CUDA 버전은 릴리스 노트에서 확인할 수 있습니다.

참고:

- EFA에서 `mpirun`을 사용하여 NCCL 애플리케이션을 실행할 때 EFA 지원 설치의 전체 경로를 다음과 같이 지정해야 합니다.

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

- 애플리케이션에서 EFA를 사용할 수 있도록 하려면 DLAMI에서 EFA 사용에서와 같이 `mpirun` 명령에 `FI_PROVIDER="efa"`를 추가합니다.

주제

- EFA 보안 그룹 준비
- 인스턴스 시작
- EFA 연결 확인

EFA 보안 그룹 준비

EFA에는 보안 그룹 자체 내의 모든 인바운드 및 아웃바운드 트래픽을 허용하는 보안 그룹이 필요합니다. 자세한 내용은 EFA 설명서를 참조하세요.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 보안 그룹을 선택한 다음, 보안 그룹 생성을 선택합니다.
3. 보안 그룹 생성 창에서 다음을 수행하세요.
    - 보안 그룹 이름의 경우 `EFA-enabled security group`과 같은 보안 그룹의 고유한 이름을 입력합니다.

- (선택 사항) 설명에 보안 그룹에 대한 간략한 설명을 입력합니다.
- VPC에서는 EFA 사용 인스턴스를 시작하려는 VPC를 선택합니다.
- 생성(Create)을 선택합니다.

4. 생성한 보안 그룹을 선택하고 설명 탭에서 그룹 ID를 복사합니다.
5. 인바운드 및 아웃바운드 탭에서 다음을 수행합니다.

   - 편집을 선택합니다.
   - 유형(Type)에서 모든 트래픽(All traffic)을 선택합니다.
   - 소스(Source)에서 사용자 지정(Custom)을 선택합니다.
   - 복사한 보안 그룹 ID를 필드에 붙여넣습니다.
   - 저장을 선택합니다.

6. Linux 인스턴스에 대한 인바운드 트래픽 권한 부여를 참조하여 인바운드 트래픽을 활성화합니다. 이 단계를 건너뛰면 DLAMI 인스턴스와 통신할 수 없습니다.

인스턴스 시작

의 EFA AWS Deep Learning AMIs 는 현재 다음 인스턴스 유형 및 운영 체제에서 지원됩니다.

- 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

다음 섹션에서는 EFA 활성화 DLAMI 인스턴스를 시작하는 방법을 설명합니다. EFA 활성화 인스턴스 실행에 대한 자세한 내용은 클러스터 배치 그룹으로 EFA 활성화 인스턴스 실행을 참조하세요.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 인스턴스 시작을 선택합니다.
3. AMI 선택 페이지에서 DLAMI 릴리스 노트 페이지에 있는 지원되는 DLAMI를 선택합니다.
4. 인스턴스 유형 선택 페이지에서 다음과 같이 지원되는 인스턴스 유형 중 하나를 선택하고 다음: 인스턴스 세부 정보 구성을 선택합니다. 지원되는 인스턴스 목록은 EFA 및 MPI 시작 링크를 참조하세요.
5. [Configure Instance Details] 페이지에서 다음을 수행합니다.

   - 인스턴스 수에 시작할 EFA를 사용한 인스턴스 수를 입력합니다.

- 네트워크 및 서브넷에서 인스턴스를 시작할 VPC와 서브넷을 선택합니다.
- [선택 사항] 배치 그룹에서 배치 그룹에 인스턴스 추가를 선택합니다. 최상의 성능을 위해 배치 그룹 내에서 인스턴스를 시작합니다.
- [선택 사항] 배치 그룹 이름에서 새 배치 그룹 추가를 선택하고 배치 그룹을 설명하는 이름을 입력한 뒤 배치 그룹 전략에서 클러스터를 선택합니다.
- 이 페이지에서 “Elastic Fabric Adapter”를 활성화해야 합니다. 이 옵션이 비활성화된 경우 선택한 인스턴스 유형을 지원하는 서브넷으로 서브넷을 변경합니다.
- 네트워크 인터페이스 항목의 디바이스 eth0에서 새 네트워크 인터페이스를 선택합니다. 하나의 기본 IPv4 주소와 하나 이상의 보조 IPv4 주소를 입력할 수도 있습니다. 연결된 IPv6 CIDR 블록이 있는 서브넷에서 인스턴스를 시작하는 경우 기본 IPv6 주소 및 하나 이상의 보조 IPv6 주소를 입력할 수도 있습니다.
- 다음: 스토리지 추가를 선택합니다.

6. 스토리지 추가 페이지에서 인스턴스를 연결할 볼륨과 AMI로 지정한 볼륨(루트 디바이스 볼륨 등)을 지정하고 다음: 태그 추가를 선택합니다.
7. 태그 추가 페이지에서 사용자에게 친숙한 이름 등의 인스턴스 태그를 지정한 후 다음: 보안 그룹 구성(Next: Configure Security Group)을 선택합니다.
8. 보안 그룹 구성 페이지의 보안 그룹 할당에서 기존 보안 그룹 선택을 선택하고 이전에 생성한 보안 그룹을 선택합니다.
9. [검토 및 시작(Review and Launch)]를 선택합니다.
10. 인스턴스 시작 검토 페이지에서 설정을 검토한 후 시작을 선택하여 키 페어를 선택하고 인스턴스를 시작합니다.

EFA 연결 확인

콘솔에서

인스턴스를 시작한 후 AWS 콘솔에서 인스턴스 세부 정보를 확인합니다. 이렇게 하려면 EC2 콘솔에서 인스턴스를 선택하고 페이지의 아래쪽 창에 있는 설명 탭을 확인합니다. ‘Network Interfaces: eth0’ 매개 변수를 찾아 eth0을 클릭하면 팝업이 열립니다. 'Elastic Fabric Adapter'가 활성화되어 있는지 확인합니다.

EFA가 활성화되지 않은 경우 다음 중 하나를 사용하여 이 문제를 해결할 수 있습니다.

- EC2 인스턴스를 종료하고 동일한 단계로 새 인스턴스를 시작합니다. EFA가 연결되어 있는지 확인합니다.

- 기존 인스턴스에 EFA를 연결합니다.

  1. EC2 콘솔에서 네트워크 인터페이스로 이동합니다.
  2. Create a Network Interface(네트워크 인터페이스 생성)를 클릭합니다.
  3. 인스턴스가 있는 서브넷과 동일한 서브넷을 선택합니다.
  4. 'Elastic Fabric Adapter'를 활성화하고 생성을 클릭합니다.
  5. EC2 인스턴스 탭으로 돌아가서 인스턴스를 선택합니다.
  6. 작업: 인스턴스 상태로 이동하여 EFA를 연결하기 전에 인스턴스를 중지합니다.
  7. 작업에서 네트워킹: Networking: Attach Network Interface(네트워크 인터페이스 연결)를 선택합니다.
  8. 방금 생성한 인터페이스를 선택하고 연결을 클릭합니다.
  9. 인스턴스를 재시작합니다.

인스턴스에서

다음 테스트 스크립트가 DLAMI에 이미 표시됩니다. 이를 실행하여 커널 모듈이 올바르게 로드되었는지 확인합니다.

```
$ fi_info -p efa
```

출력은 다음과 비슷한 형태가 됩니다.

```
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
```

보안 그룹 구성 확인

다음 테스트 스크립트가 DLAMI에 이미 표시됩니다. 이를 실행하여 생성한 보안 그룹이 올바르게 구성되었는지 확인합니다.

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

출력은 다음과 비슷한 형태가 됩니다.

```
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
```

응답이 중단되거나 작업이 완료되지 않으면 보안 그룹에 올바른 인바운드/아웃바운드 규칙이 있는지 확인합니다.

DLAMI에서 EFA 사용

다음 섹션에서는 EFA를 사용하여 AWS Deep Learning AMIs에서 다중 노드 애플리케이션을 실행하는 방법에 대해 설명합니다.

EFA를 사용하여 다중 노드 애플리케이션 실행

노드 클러스터에서 애플리케이션을 실행하려면 다음 구성이 필요합니다.

주제

- 암호 없는 SSH 사용
- 호스트 파일 생성
- NCCL 테스트

암호 없는 SSH 사용

클러스터의 노드 하나를 리더 노드로 선택합니다. 나머지 노드들을 멤버 노드라고 합니다.

1. 리더 노드에서 RSA 키 페어를 생성합니다.

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

2. 리더 노드에서 프라이빗 키의 사용 권한을 변경합니다.

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

3. 퍼블릭 키를 ~/.ssh/id_rsa.pub에 복사하고 클러스터 멤버 노드의 ~/.ssh/authorized_keys에 추가합니다.
4. 이제 프라이빗 IP를 사용하여 리더 노드에서 멤버 노드에 직접 로그인 할 수 있습니다.

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

5. 리더 노드에서 ~ /.ssh/config 파일을 추가하여 strictHostKeyChecking을 비활성화하고 리더 노드에서 에이전트 전달을 활성화합니다.

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

6. Amazon Linux 2 인스턴스의 리더 노드에서 다음 명령을 실행하여 구성 파일에 올바른 권한을 제공합니다.

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

호스트 파일 생성

리더 노드에서 클러스터의 노드를 식별하기 위한 호스트 파일을 생성합니다. 호스트 파일에는 클러스터의 각 노드에 대한 항목이 있어야 합니다. 파일 ~/hosts를 생성하고 다음과 같이 프라이빗 IP를 사용하여 각 노드를 추가합니다.

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

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

NCCL 테스트

> **Note**
>
> 이러한 테스트는 EFA 버전 1.38.0 및 OFI NCCL 플러그인 1.13.2를 사용하여 실행되었습니다.

다음은 여러 컴퓨팅 노드의 기능과 성능을 모두 테스트하도록 Nvidia에서 제공하는 NCCL 테스트의 하위 집합입니다.

지원되는 인스턴스: P3dn, P4, P5, P5e, P5en

성능 테스트

P4d.24xlarge에 대한 다중 노드 NCCL 성능 테스트

EFA에서 NCCL 성능을 확인하려면 공식 NCCL-Tests Repo에서 사용할 수 있는 표준 NCCL 성능 테스트를 실행합니다. DLAMI는 CUDA XX.X용으로 이미 구축된 이 테스트와 함께 제공됩니다. 마찬가지로 EFA를 사용하여 자체 스크립트를 실행할 수 있습니다.

자체 스크립트를 작성할 때 다음 지침을 참조하세요.

- EFA에서 NCCL 애플리케이션을 실행하는 동안 예제에서와 같이 mpirun으로의 전체 경로를 사용합니다.
- 클러스터의 인스턴스 및 GPU 개수에 따라 매개 변수 np와 N을 변경합니다.
- NCCL_DEBUG=INFO 플래그를 추가하고 로그에 EFA 사용법이 "Selected Provider is "EFA로 표시되어 있는지 확인합니다.
- 검증을 위해 구문 분석할 훈련 로그 위치 설정

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

멤버 노드에서 명령 `watch nvidia-smi`을 사용하여 GPU 사용을 모니터링합니다. 다음 `watch nvidia-smi` 명령은 일반 CUDA xx.x 버전에서만 사용할 수 있으며 인스턴스의 운영 체제에 따라 달라질 수 있습니다. 스크립트에서 CUDA 버전을 교체하여 Amazon EC2 인스턴스에서 사용 가능한 모든 CUDA 버전에 대해 명령을 실행할 수 있습니다.

- Amazon Linux 2, Amazon Linux 2023:

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

- Ubuntu 20.04, Ubuntu 20.04:

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

출력은 다음과 같아야 합니다.

```
# nThread 1 nGpus 1 minBytes 8 maxBytes 1073741824 step: 2(factor) warmup iters: 5
 iters: 100 agg iters: 1 validation: 1 graph: 0
#
# Using devices
#  Rank  0 Group  0 Pid  33378 on ip-172-31-42-25 device  0 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  1 Group  0 Pid  33379 on ip-172-31-42-25 device  1 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  2 Group  0 Pid  33380 on ip-172-31-42-25 device  2 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank  3 Group  0 Pid  33381 on ip-172-31-42-25 device  3 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank  4 Group  0 Pid  33382 on ip-172-31-42-25 device  4 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank  5 Group  0 Pid  33383 on ip-172-31-42-25 device  5 [0x90] NVIDIA A100-
SXM4-40GB
```

```
#  Rank  6 Group  0 Pid  33384 on ip-172-31-42-25 device  6 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank  7 Group  0 Pid  33385 on ip-172-31-42-25 device  7 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank  8 Group  0 Pid  30378 on ip-172-31-43-8 device  0 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  9 Group  0 Pid  30379 on ip-172-31-43-8 device  1 [0x10] NVIDIA A100-SXM4-40GB
#  Rank 10 Group  0 Pid  30380 on ip-172-31-43-8 device  2 [0x20] NVIDIA A100-SXM4-40GB
#  Rank 11 Group  0 Pid  30381 on ip-172-31-43-8 device  3 [0x20] NVIDIA A100-SXM4-40GB
#  Rank 12 Group  0 Pid  30382 on ip-172-31-43-8 device  4 [0x90] NVIDIA A100-SXM4-40GB
#  Rank 13 Group  0 Pid  30383 on ip-172-31-43-8 device  5 [0x90] NVIDIA A100-SXM4-40GB
#  Rank 14 Group  0 Pid  30384 on ip-172-31-43-8 device  6 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank 15 Group  0 Pid  30385 on ip-172-31-43-8 device  7 [0xa0] NVIDIA A100-SXM4-40GB
ip-172-31-42-25:33385:33385 [7] NCCL INFO cudaDriverVersion 12060
ip-172-31-43-8:30383:30383 [5] NCCL INFO Bootstrap : Using ens32:172.31.43.8
ip-172-31-43-8:30383:30383 [5] NCCL INFO NCCL version 2.23.4+cuda12.5
...
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.13.2-aws
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Using Libfabric version 1.22
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Using CUDA driver version 12060 with
 runtime 12050
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting NCCL_NVLSTREE_MAX_CHUNKSIZE
 to 512KiB
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Setting NCCL_NVLS_CHUNKSIZE to 512KiB
ip-172-31-42-25:33384:33451 [6] NCCL INFO NET/OFI Running on p4d.24xlarge platform,
 Setting NCCL_TOPO_FILE environment variable to /opt/amazon/ofi-nccl/share/aws-ofi-
nccl/xml/p4d-24xl-topo.xml
...
-----------------------------some output truncated-----------------------------------
#                                                              out-of-place
          in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong
  time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)
  (us)  (GB/s)  (GB/s)
           8             2     float     sum      -1    180.3    0.00    0.00      0
 179.3    0.00    0.00      0
          16             4     float     sum      -1    178.1    0.00    0.00      0
 177.6    0.00    0.00      0
          32             8     float     sum      -1    178.5    0.00    0.00      0
 177.9    0.00    0.00      0
```

```
          64             16     float     sum      -1    178.8    0.00    0.00       0
178.7    0.00    0.00       0
         128             32     float     sum      -1    178.2    0.00    0.00       0
177.8    0.00    0.00       0
         256             64     float     sum      -1    178.6    0.00    0.00       0
178.8    0.00    0.00       0
         512            128     float     sum      -1    177.2    0.00    0.01       0
177.1    0.00    0.01       0
        1024            256     float     sum      -1    179.2    0.01    0.01       0
179.3    0.01    0.01       0
        2048            512     float     sum      -1    181.3    0.01    0.02       0
181.2    0.01    0.02       0
        4096           1024     float     sum      -1    184.2    0.02    0.04       0
183.9    0.02    0.04       0
        8192           2048     float     sum      -1    191.2    0.04    0.08       0
190.6    0.04    0.08       0
       16384           4096     float     sum      -1    202.5    0.08    0.15       0
202.3    0.08    0.15       0
       32768           8192     float     sum      -1    233.0    0.14    0.26       0
232.1    0.14    0.26       0
       65536          16384     float     sum      -1    238.6    0.27    0.51       0
235.1    0.28    0.52       0
      131072          32768     float     sum      -1    237.2    0.55    1.04       0
236.8    0.55    1.04       0
      262144          65536     float     sum      -1    248.3    1.06    1.98       0
247.0    1.06    1.99       0
      524288         131072     float     sum      -1    309.2    1.70    3.18       0
307.7    1.70    3.20       0
     1048576         262144     float     sum      -1    408.7    2.57    4.81       0
404.3    2.59    4.86       0
     2097152         524288     float     sum      -1    613.5    3.42    6.41       0
607.9    3.45    6.47       0
     4194304        1048576     float     sum      -1    924.5    4.54    8.51       0
914.8    4.58    8.60       0
     8388608        2097152     float     sum      -1   1059.5    7.92   14.85       0
1054.3    7.96   14.92       0
    16777216        4194304     float     sum      -1   1269.9   13.21   24.77       0
1272.0   13.19   24.73       0
    33554432        8388608     float     sum      -1   1642.7   20.43   38.30       0
1636.7   20.50   38.44       0
    67108864       16777216     float     sum      -1   2446.7   27.43   51.43       0
2445.8   27.44   51.45       0
   134217728       33554432     float     sum      -1   4143.6   32.39   60.73       0
4142.4   32.40   60.75       0
```

```
   268435456      67108864     float     sum      -1   7351.9   36.51   68.46      0
 7346.7   36.54   68.51      0
   536870912     134217728     float     sum      -1    13717   39.14   73.39      0
 13703   39.18   73.46      0
  1073741824     268435456     float     sum      -1    26416   40.65   76.21      0
 26420   40.64   76.20      0
...
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 15.5514
```

검증 테스트

EFA 테스트가 유효한 결과를 반환했는지 검증하려면 다음 테스트를 사용하여 확인합니다.

- EC2 인스턴스 메타데이터를 사용하여 인스턴스 유형 가져오기:

```
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)
```

- 성능 테스트 실행
- 다음 파라미터 설정

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- 다음과 같이 결과 검증:

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

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

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

    # Validation of logs
```

```
    grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-
specific options text not found"; exit 1; }
    grep "busbw" ${TRAINING_LOG} || { echo "busbw text not found"; exit 1; }
    grep "Avg bus bandwidth " ${TRAINING_LOG} || { echo "Avg bus bandwidth text not
 found"; exit 1; }
    grep "NCCL version $NCCL_VERSION" ${TRAINING_LOG} || { echo "Text not found: NCCL
 version $NCCL_VERSION"; exit 1; }
    if [[ ${INSTANCE_TYPE} == "p4d.24xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found: NET/
Libfabric/0/GDRDMA"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5e.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5en.48xlarge" ]]; then
        grep "NET/Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus bandwidth
 text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 16 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p3dn.24xlarge" ]]; then
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
    fi
    echo "***************************** check_efa_nccl_all_reduce passed for cuda
 version ${CUDA_VERSION} *****************************"
else
    echo "***************************** check_efa_nccl_all_reduce failed for cuda
 version ${CUDA_VERSION} *****************************"
fi
```

- 벤치마크 데이터에 액세스하기 위해 다중 노드 all_reduce 테스트에서 테이블 출력의 마지막 행을 구문 분석할 수 있습니다.

```
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 모니터링 및 최적화

다음 단원에서는 GPU 최적화 및 모니터링 옵션을 안내합니다. 이 단원은 모니터링, 감독, 사전 처리 및 교육의 일반적인 워크플로우처럼 구성되어 있습니다.

- 모니터링
  - CloudWatch를 이용한 GPU 모니터링
- 최적화
  - 사전 처리
  - 학습

모니터링

DLAMI는 몇 가지 GPU 모니터링 도구가 사전 설치된 상태로 제공됩니다. 이 설명서에서는 다운로드 및 설치에 사용할 수 있는 도구에 대해서도 설명합니다.

- CloudWatch를 이용한 GPU 모니터링 - Amazon CloudWatch에 GPU 사용 통계를 보고하는 사전 설치된 유틸리티입니다.
- nvidia-smi CLI - 전반적인 GPU 컴퓨팅 및 메모리 사용률을 모니터링하는 유틸리티입니다. 이는 AWS Deep Learning AMIs (DLAMI)에 사전 설치되어 있습니다.
- NVML C 라이브러리 - GPU 모니터링 및 관리 기능에 직접 액세스할 수 있는 C 기반 API입니다. 이 API는 후드 아래에서 nvidia-smi CLI에 사용되며 DLAMI에 사전 설치되어 있습니다. 또한 Python 및 Perl 바인딩을 가지고 있어 이러한 언어로 신속하게 개발이 가능합니다. DLAMI에 사전 설치된 gpumon.py 유틸리티는 nvidia-ml-py에서 pynvml 패키지를 사용합니다.

- NVIDIA DCGM - 클러스터 관리 도구입니다. 이 도구를 설치하고 구성하는 방법을 알아보려면 개발자 페이지를 방문하세요.

> **Tip**
>
> DLAMI에 설치된 CUDA 도구를 사용하는 방법에 대한 최신 정보는 NVIDIA의 개발자 블로그에서 확인하세요.
>
> - Nsight IDE 및 nvprof를 사용한 TensorCore 사용률 모니터링.

CloudWatch를 이용한 GPU 모니터링

GPU에서 DLAMI를 사용할 때는 교육 또는 추론 동안 사용량을 추적하는 방법을 찾고 있는 경우일 수 있습니다. 이는 데이터 파이프라인을 최적화하고 딥 러닝 네트워크를 튜닝할 때 유용할 수 있습니다.

CloudWatch GPU 지표 구성 방법은 두 가지입니다.

- AWS CloudWatch 에이전트를 사용하여 지표 구성(권장)
- 사전 설치된 `gpumon.py` 스크립트로 지표를 구성합니다.

AWS CloudWatch 에이전트를 사용하여 지표 구성(권장)

DLAMI를 CloudWatch 에이전트와 통합하여 Amazon EC2 가속화 인스턴스에서 GPU 지표를 구성하고 GPU 공동 프로세스의 사용률을 모니터링할 수 있습니다.

DLAMI를 사용하여 GPU 지표를 구성하는 방법은 네 가지입니다.

- GPU 지표 최소 구성
- GPU 지표 부분 구성
- 사용 가능한 GPU 지표 전체 구성
- 사용자 지정 GPU 지표 구성

업데이트 및 보안 패치에 관한 자세한 내용은 AWS CloudWatch 에이전트에 대한 보안 패치 적용를 참조하세요.

사전 조건

시작하기 전에 인스턴스가 지표를 CloudWatch로 푸시할 수 있도록 Amazon EC2 인스턴스 IAM 권한을 구성해야 합니다. 자세한 구성 방법은 CloudWatch 에이전트와 함께 사용하기 위한 IAM 역할 및 사용자 생성을 참조하세요.

GPU 지표 최소 구성

`dlami-cloudwatch-agent@minimal` systemd 서비스를 사용하여 GPU 최소 지표를 구성합니다. 이 서비스의 지표 구성은 다음과 같습니다.

- `utilization_gpu`
- `utilization_memory`

GPU 지표 최소 사전 구성에 대한 systemd 서비스는 다음 위치에서 찾을 수 있습니다.

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

다음 명령을 사용하여 systemd 서비스를 활성화하고 시작합니다.

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

GPU 지표 부분 구성

`dlami-cloudwatch-agent@partial` systemd 서비스를 사용하여 GPU 부분 지표를 구성합니다. 이 서비스의 지표 구성은 다음과 같습니다.

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

GPU 지표 부분 사전 구성에 대한 systemd 서비스는 다음 위치에서 찾을 수 있습니다.

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

다음 명령을 사용하여 systemd 서비스를 활성화하고 시작합니다.

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

사용 가능한 GPU 지표 전체 구성

dlami-cloudwatch-agent@all systemd 서비스를 사용하여 GPU 전체 지표를 구성합니다. 이 서비스의 지표 구성은 다음과 같습니다.

- 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

GPU 전체 지표 사전 구성에 대한 systemd 서비스는 다음 위치에서 찾을 수 있습니다.

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

다음 명령을 사용하여 systemd 서비스를 활성화하고 시작합니다.

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

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

사용자 지정 GPU 지표 구성

사전 구성된 지표가 요구 사항을 충족하지 않는 경우 사용자 지정 CloudWatch 에이전트 구성 파일을 생성할 수 있습니다.

사용자 지정 구성 파일 생성

사용자 지정 구성 파일을 생성하려면 수동으로 CloudWatch 에이전트 구성 파일 생성 또는 편집에 있는 세부 단계를 참조하세요.

이 예제에서는 스키마 정의 위치를 `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`이라고 가정합니다.

사용자 지정 파일로 지표 구성

다음 명령을 실행하여 사용자 지정 파일에 따라 CloudWatch 에이전트를 구성합니다.

```
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
```

AWS CloudWatch 에이전트에 대한 보안 패치 적용

새로 릴리스된 DLAMIs는 사용 가능한 최신 AWS CloudWatch 에이전트 보안 패치로 구성됩니다. 선택한 운영 체제에 따라 현재 DLAMI를 최신 보안 패치로 업데이트하려면 다음 섹션을 참조하세요.

Amazon Linux 2

yum를 사용하여 Amazon Linux 2 DLAMI에 대한 최신 AWS CloudWatch 에이전트 보안 패치를 가져옵니다.

```
 sudo yum update
```

Ubuntu

Ubuntu를 사용하는 DLAMI에 대한 최신 AWS CloudWatch 보안 패치를 가져오려면 Amazon S3 다운로드 링크를 사용하여 AWS CloudWatch 에이전트를 다시 설치해야 합니다.

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

Amazon S3 다운로드 링크를 사용하여 AWS CloudWatch 에이전트를 설치하는 방법에 대한 자세한 내용은 서버에 CloudWatch 에이전트 설치 및 실행을 참조하세요.

사전 설치된 **gpumon.py** 스크립트로 지표를 구성합니다.

gpumon.py라는 유틸리티는 DLAMI에 사전 설치되어 있습니다. 이 유틸리티는 CloudWatch에 통합되어 GPU당 사용량(GPU 메모리, GPU 온도 및 GPU 전력)의 모니터링을 지원합니다. 스크립트가 CloudWatch에 모니터링된 데이터를 정기적으로 전송합니다. 스크립트에서 몇 가지 설정을 변경하여 CloudWatch에 전송 중인 데이터에 대한 세부 수준을 구성할 수 있습니다. 하지만 스크립트를 시작하기 전에 지표를 수신하도록 CloudWatch를 설정해야 합니다.

CloudWatch를 이용해 GPU 모니터링을 설정 및 실행하는 방법

1. IAM 사용자를 생성하거나 기존 사용자를 수정하여 CloudWatch에 지표를 게시하기 위한 정책을 수립합니다. 새로 사용자를 생성하는 경우에는 다음 단계에서 필요할 수 있기 때문에 자격 증명을 적어두십시오.

   검색할 IAM 정책은 “cloudwatch:PutMetricData”입니다. 추가된 정책은 다음과 같습니다.

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

   > **Tip**
   >
   > IAM 사용자를 생성하고 CloudWatch에서 정책을 추가하는 방법에 대한 자세한 내용은 CloudWatch 설명서를 참조하세요.

2. DLAMI에서 AWS 구성을 실행하고 IAM 사용자 자격 증명을 지정합니다.

   ```
   $ aws configure
   ```

3. 실행에 앞서 gpumon 유틸리티를 약간 수정해야 할 수 있습니다. 다음 코드 블록에 명시된 위치에서 gpumon 유틸리티 및 README를 찾을 수 있습니다. gpumon.py 스크립트에 대한 자세한 내용은 스크립트의 Amazon S3 위치를 참조하세요.

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

옵션:

- 인스턴스가 us-east-1에 없는 경우에는 gpumon.py에서 리전을 변경합니다.
- CloudWatch namespace 또는 store_reso의 보고 기간 같은 다른 파라미터를 변경합니다.

4. 현재, 스크립트는 Python 3만 지원합니다. 선호하는 프레임워크의 Python 3 환경을 활성화하거나 DLAMI의 일반적인 Python 3 환경을 활성화합니다.

```
$ source activate python3
```

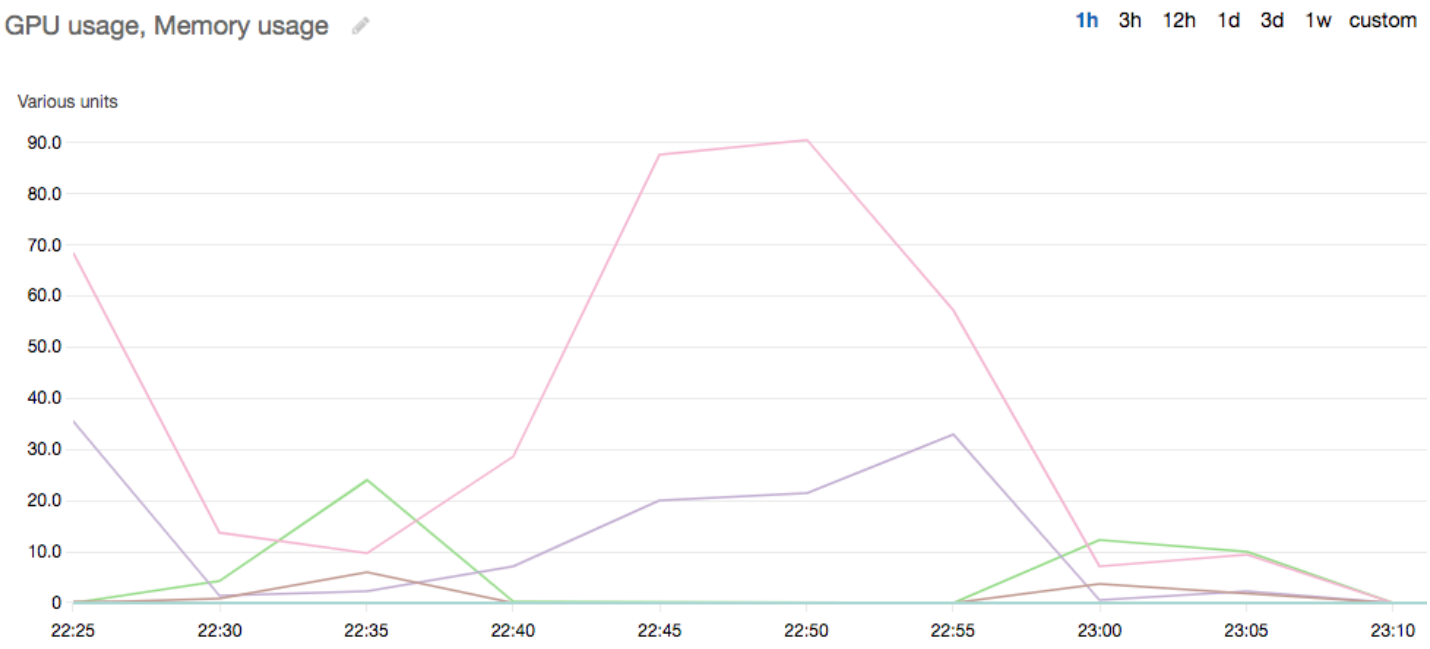
5. 배경에서 gpumon 유틸리티를 실행합니다.

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

6. 브라우저를 열어 https://console.aws.amazon.com/cloudwatch/로 이동한 다음 지표를 선택합니다. 네임스페이스는 'DeepLearningTrain'이 됩니다.

> **Tip**
>
> gpumon.py를 수정하여 네임스페이스를 변경할 수 있습니다. 또한 store_reso을 조정하여 보고 간격을 수정할 수도 있습니다.

다음은 gpumon.py를 실행하여 p2.8xlarge 인스턴스에서 교육 작업을 모니터링할 때 CloudWatch 차트 보고의 예제입니다.



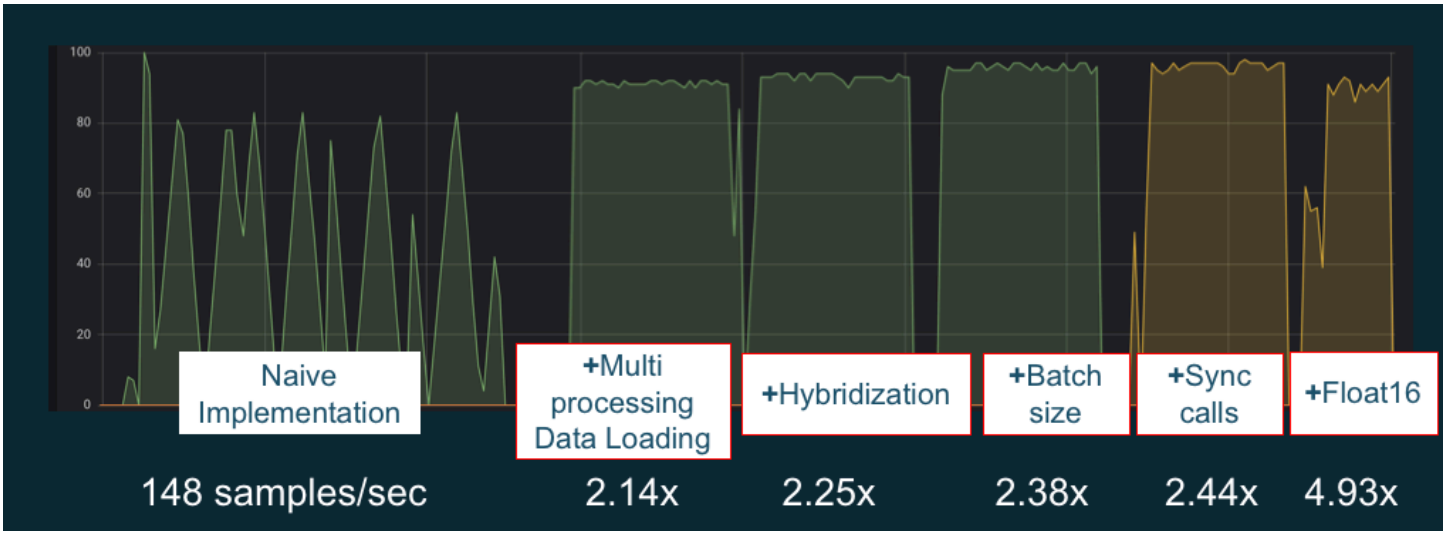
GPU 모니터링 및 최적화에 대한 이러한 기타 주제들에 관심이 있을 수 있습니다.

- 모니터링
  - CloudWatch를 이용한 GPU 모니터링
- 최적화
  - 사전 처리
  - 학습

최적화

GPU를 최대한 활용하기 위해 데이터 파이프라인을 최적화하고 딥 러닝 네트워크를 튜닝할 수 있습니다. 아래 차트에서 설명하듯, 단순하고 기본적인 방식으로 구현된 신경망에서는 GPU를 일관성 있게 사용하지 못해서 잠재력이 충분히 발휘되지 못할 수 있습니다. 사전 처리 및 데이터 로딩을 최적화하면 CPU에서 GPU로의 병목을 줄일 수 있습니다. 하이브리드화를 사용(프레임워크에서 지원할 때)하여 배치 크기를 조정하고, 호출을 동기화하여 신경망 자체를 조정할 수 있습니다. 또한 대부분 프레임워크에서 다중 정밀도(float16 또는 int8) 교육을 사용하여 처리량 개선에 획기적인 영향을 미칠 수도 있습니다.

다음 차트에는 서로 다른 최적화를 적용할 때 누적되는 성능상 이점을 보여줍니다. 처리 중인 데이터와 최적화 중인 네트워크에 따라 결과는 달라집니다.



GPU 성능 최적화 예 차트 소스: MXNet Gluon에서 성능 개선 요령

다음 설명서는 DLAMI에서 실행 가능하고 GPU 성능을 높이는 데 도움을 주는 옵션에 대해 설명합니다.

주제

- 사전 처리
- 학습

사전 처리

변환 또는 증강을 통한 데이터 사전 처리는 CPU 바운드 프로세스인 경우가 종종 있기 때문에 전체 파이프라인에서 병목이 될 수 있습니다. 프레임워크에는 이미지 처리를 위한 연산자가 내장되어 있지만, DALI(Data Augmentation Library)는 프레임워크에 내장된 옵션을 통한 성능 개선을 입증합니다.

- NVIDIA DALI(Data Augmentation Library): DALI는 GPU에 데이터 증강을 오프로드합니다. DLAMI에 사전 설치되어 있지 않지만, 이를 설치하거나 지원되는 프레임워크 컨테이너를 DLAMI나 다른 Amazon Elastic Compute Cloud 인스턴스에 로드하여 액세스할 수 있습니다. 자세한 내용은 NVIDIA 웹 사이트의 DALI 프로젝트 페이지를 참조하세요. 예제 사용 사례와 코드 샘플을 다운로드하려면 SageMaker 사전 처리 교육 성능 샘플을 참조하세요.
- nvJPEG: C 프로그래머를 위한 GPU 기반의 JPEG 디코더 라이브러리입니다. 단일 이미지 또는 배치를 비롯해 딥 러닝에서 일반적인 후속 변환 작업을 디코딩할 수 있도록 지원합니다. nvJPEG에는 DALI가 내장되어 있을 수도 있고, NVIDIA 웹 사이트의 nvjpeg 페이지에서 다운로드하여 별도로 사용할 수 있습니다.

GPU 모니터링 및 최적화에 대한 이러한 기타 주제들에 관심이 있을 수 있습니다.

- 모니터링
  - CloudWatch를 이용한 GPU 모니터링
- 최적화
  - 사전 처리
  - 학습

학습

혼합 정밀도 교육을 통해 같은 양의 메모리로 더 큰 네트워크를 배포하거나 단일 또는 이중 정밀도 네트워크와 비교해 메모리 사용량을 줄여서 컴퓨팅 성능을 높일 수 있습니다. 또한 보다 소규모로 더 빨리 데이터를 전송할 수 있다는 이점이 있는데, 이는 여러 노드에 분산된 교육에서 중요한 요소입니다. 혼합 정밀도 교육을 활용하려면 데이터 캐스팅 및 손실을 조정해야 합니다. 다음은 혼합 정밀도를 지원하는 프레임워크에서 이를 수행하는 방법을 설명하는 설명서입니다.

- NVIDIA 딥 러닝 SDK - MXNet, PyTorch 및 TensorFlow를 위한 혼합 정밀도 구현을 설명하는 NVIDIA 웹 사이트의 문서.

> **Tip**
>
> 선택한 프레임워크를 위한 웹 사이트를 확인하고 최신 최적화 기법에 대한 "혼합 정밀도" 또는 "fp16"을 검색하세요. 다음과 같이 몇 가지 혼합 정밀도 설명서가 도움이 될 수 있습니다.
>
> - TensorFlow를 이용한 혼합 정밀도 교육(비디오) - NVIDIA 블로그 사이트에서 제공.
> - MXNet에서 float16을 사용한 혼합 정밀도 교육 - MXNet 웹 사이트의 FAQ 문서.
> - NVIDIA Apex: PyTorch를 이용한 간편한 혼합 정밀도 교육을 위한 도구 - NVIDIA 웹 사이트의 블로그 기사.

GPU 모니터링 및 최적화에 대한 이러한 기타 주제들에 관심이 있을 수 있습니다.

- 모니터링
  - CloudWatch를 이용한 GPU 모니터링
- 최적화
  - 사전 처리

- 학습

DLAMI를 사용하는 AWS Inferentia 칩

AWS Inferentia는 고성능 추론 예측에 사용할 수 AWS 있도록에서 설계된 사용자 지정 기계 학습 칩입니다. 칩을 사용하려면 Amazon Elastic Compute Cloud 인스턴스를 설정하고 AWS Neuron 소프트웨어 개발 키트(SDK)를 사용하여 Inferentia 칩을 호출합니다. Inferentia를 최적으로 사용할 수 있도록 DLAMI(AWS Deep Learning AMIs)에 Neuron이 내장되어 있습니다.

다음 섹션에서는 DLAMI에서 Inferentia를 활용하는 방법을 설명합니다.

내용

- AWS Neuron을 사용하여 DLAMI 인스턴스 시작
- AWS Neuron에서 DLAMI 사용

AWS Neuron을 사용하여 DLAMI 인스턴스 시작

최신 DLAMI는 AWS Inferentia와 함께 사용할 준비가 되었으며 AWS Neuron API 패키지와 함께 제공됩니다. DLAMI 인스턴스를 시작하려면 DLAMI 시작 및 구성을 참조하세요. DLAMI가 있으면 여기의 단계를 사용하여 AWS Inferentia 칩과 AWS Neuron 리소스가 활성 상태인지 확인합니다.

내용

- 인스턴스 확인
- AWS Inferentia 디바이스 식별
- 리소스 사용량 보기
- Neuron Monitor 사용(neuron-monitor)
- Neuron 소프트웨어 업그레이드

인스턴스 확인

인스턴스를 사용하기 전에 인스턴스가 제대로 설정되고 Neuron으로 구성되어 있는지 확인합니다.

AWS Inferentia 디바이스 식별

인스턴스의 Inferentia 디바이스 수를 식별하려면 다음 명령을 사용합니다.

```
neuron-ls
```

인스턴스에 Inferentia 디바이스가 연결되어 있는 경우 다음과 유사하게 출력됩니다.

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

제공된 출력은 INF1.6xlarge 인스턴스에서 가져온 것이며 다음 열을 포함합니다.

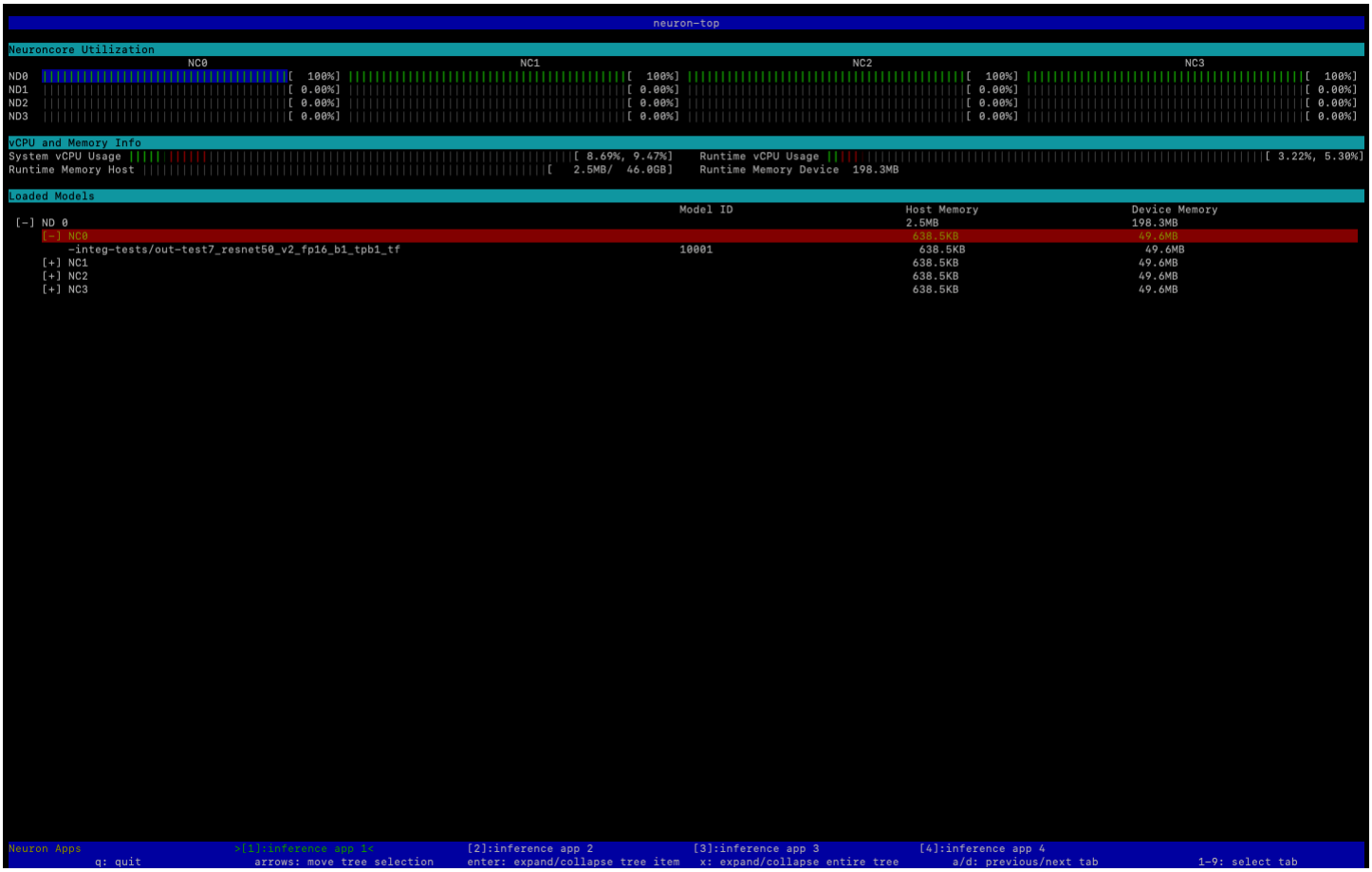
- NEURON DEVICE: NeuronDevice에 할당된 논리적 ID입니다. 이 ID는 여러 NeuronDevices를 사용하도록 다중 런타임을 구성할 때 사용됩니다.
- NEURON CORES: NeuronDevice에 존재하는 NeuronCores의 수입니다.
- NEURON MEMORY: NeuronDevice에 있는 DRAM 메모리의 양입니다.
- CONNECTED DEVICES: NeuronDevice에 연결된 다른 NeuronDevice입니다.,
- PCI BDF: NeuronDevice의 PCI 버스 디바이스 함수(BDF) ID입니다.

리소스 사용량 보기

`neuron-top` 명령을 사용하여 NeuronCore 및 vCPU 사용률, 메모리 사용량, 로드된 모델 및 Neuron 애플리케이션에 대한 유용한 정보를 볼 수 있습니다. 인수 없이 `neuron-top`를 실행하면 NeuronCores를 사용하는 모든 기계 학습 응용 프로그램의 데이터가 표시됩니다.

```
neuron-top
```

애플리케이션이 NeuronCore 4개를 사용하는 경우 다음과 유사하게 출력됩니다.



Neuron 기반 추론 애플리케이션을 모니터링하고 최적화하기 위한 리소스에 대한 자세한 내용은 Neuron 도구를 참조하세요.

Neuron Monitor 사용(neuron-monitor)

Neuron Monitor는 시스템에서 실행되는 Neuron 런타임에서 지표를 수집하고 수집된 데이터를 JSON 형식의 stdout에 스트리밍합니다. 이러한 지표는구성 파일을 제공하여 구성하는 지표 그룹으로 구성됩니다. Neuron Monitor에 대한 자세한 내용은 Neuron Monitor사용 설명서를 참조하세요.

Neuron 소프트웨어 업그레이드

DLAMI 내에서 Neuron SDK 소프트웨어를 업데이트하는 방법에 대한 자세한 내용은 AWS Neuron Setup Guide를 참조하세요.

다음 단계

AWS Neuron에서 DLAMI 사용

AWS Neuron에서 DLAMI 사용

AWS Neuron SDK를 사용하는 일반적인 워크플로는 컴파일 서버에서 이전에 훈련된 기계 학습 모델을 컴파일하는 것입니다. 그런 다음 실행을 위해 아티팩트를 Inf1 인스턴스에 배포합니다. AWS Deep Learning AMIs (DLAMI)에는 Inferentia를 사용하는 Inf1 인스턴스에서 추론을 컴파일하고 실행하는 데 필요한 모든 것이 사전 설치되어 있습니다.

다음 섹션에서는 Inferentia DLAMI를 사용하는 방법을 설명합니다.

내용

TensorFlow-Neuron 및 AWS Neuron 컴파일러 사용

이 자습서에서는 AWS Neuron 컴파일러를 사용하여 Keras ResNet-50 모델을 컴파일하고 SavedModel 형식으로 저장된 모델로 내보내는 방법을 보여줍니다. 이 형식은 일반 TensorFlow 모델에서 서로 바꿔 사용할 수 있는 형식입니다. 또한 예제 입력을 사용하여 Inf1 인스턴스에서 추론을 실행하는 방법을 알아봅니다.

Neuron SDK에 대한 자세한 내용은 AWS Neuron SDK 설명서를 참조하세요.

내용

사전 조건

이 자습서를 사용하기 전에 AWS Neuron을 사용하여 DLAMI 인스턴스 시작의 설정 단계를 완료해야 합니다. 또한 딥 러닝 및 DLAMI 사용에 익숙해야 합니다.

Conda 환경 활성화

다음 명령을 사용하여 TensorFlow-Neuron conda 환경을 활성화합니다.

```
source activate aws_neuron_tensorflow_p36
```

현재 conda 환경을 종료하려면 다음 명령을 실행합니다.

```
source deactivate
```

Resnet50 컴파일

다음 콘텐츠를 가진 **tensorflow_compile_resnet50.py**라는 Python 스크립트를 생성합니다. 이 Python 스크립트는 Keras ResNet50 모델을 컴파일하고 저장된 모델로 내보냅니다.

```
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')
```

다음 명령을 사용하여 모델을 컴파일합니다.

```
python tensorflow_compile_resnet50.py
```

컴파일 프로세스는 몇 분 정도 걸립니다. 완료되면 출력은 다음과 같아야 합니다.

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

컴파일 후 저장된 모델은 **ws_resnet50/resnet50_neuron.zip**에서 압축됩니다. 다음 명령을 사용하여 모델의 압축을 풀고 추론을 위해 샘플 이미지를 다운로드합니다.

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

ResNet50 추론

다음 콘텐츠를 가진 **tensorflow_infer_resnet50.py**라는 Python 스크립트를 생성합니다. 이 스크립트는 이전에 컴파일된 추론 모델을 사용하여 다운로드한 모델에 대한 추론을 실행합니다.

```
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])
```

다음 명령을 사용하여 모델에 대한 추론을 실행합니다.

```
python tensorflow_infer_resnet50.py
```

출력은 다음과 같아야 합니다.

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

다음 단계

AWS Neuron TensorFlow Serving 사용

AWS Neuron TensorFlow Serving 사용

이 자습서에서는 TensorFlow Serving에 사용할 저장된 모델을 내보내기 전에 그래프를 구성하고 AWS Neuron 컴파일 단계를 추가하는 방법을 보여줍니다. TensorFlow Serving은 네트워크를 통해 추론을

확장할 수 있는 지원 시스템입니다. Neuron TensorFlow Serving은 일반적인 TensorFlow Serving과 동일한 API를 사용합니다. 유일한 차이점은 저장된 모델을 AWS Inferentia용으로 컴파일해야 하며 진입점은 라는 다른 바이너리라는 것입니다`tensorflow_model_server_neuron`. 이진 파일은 `/usr/local/bin/tensorflow_model_server_neuron`에 있으며 DLAMI에 사전 설치되어 있습니다.

Neuron SDK에 대한 자세한 내용은 AWS Neuron SDK 설명서를 참조하세요.

내용

사전 조건

이 자습서를 사용하기 전에 AWS Neuron을 사용하여 DLAMI 인스턴스 시작의 설정 단계를 완료해야 합니다. 또한 딥 러닝 및 DLAMI 사용에 익숙해야 합니다.

Conda 환경 활성화

다음 명령을 사용하여 TensorFlow-Neuron conda 환경을 활성화합니다.

```
source activate aws_neuron_tensorflow_p36
```

현재 conda 환경을 종료해야 하는 경우 다음을 실행합니다.

```
source deactivate
```

저장된 모델 컴파일 및 내보내기

다음 콘텐츠를 통해 `tensorflow-model-server-compile.py` 이름으로 Python 스크립트를 생성합니다. 이 스크립트는 그래프를 구성하고 Neuron을 사용하여 컴파일합니다. 그런 다음 컴파일된 그래프를 저장된 모델로 내보냅니다.

```
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)
```

다음 명령을 사용하여 모델을 컴파일합니다.

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

출력은 다음과 같아야 합니다.

```
...
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
```

저장된 모델 제공

모델이 컴파일되면 다음 명령을 사용하여 저장된 모델을 tensorflow_model_server_neuron 이진 파일로 제공할 수 있습니다.

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

출력은 다음과 같아야 합니다. 컴파일된 모델은 추론을 준비하기 위해 서버에 의해 Inferentia 디바이스의 DRAM에 준비됩니다.

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

모델 서버에 대한 추론 요청 생성

다음 콘텐츠를 통해 `tensorflow-model-server-infer.py`라는 Python 스크립트를 생성합니다. 이 스크립트는 서비스 프레임워크인 gRPC를 통해 추론을 실행합니다.

```
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))
```

다음 명령에서 gRPC를 사용하여 모델에 대한 추론을 실행합니다.

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

출력은 다음과 같아야 합니다.

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

MXNet-Neuron 및 AWS Neuron 컴파일러 사용

MXNet-Neuron 컴파일 API는 AWS Inferentia 디바이스에서 실행할 수 있는 모델 그래프를 컴파일하는 방법을 제공합니다.

이 예에서는 API를 사용하여 ResNet-50 모델을 컴파일하고 추론을 실행하는 데 사용합니다.

Neuron SDK에 대한 자세한 내용은 AWS Neuron SDK 설명서를 참조하세요.

내용

- 사전 조건
- Conda 환경 활성화
- Resnet50 컴파일
- ResNet50 추론

사전 조건

이 자습서를 사용하기 전에 AWS Neuron을 사용하여 DLAMI 인스턴스 시작의 설정 단계를 완료해야 합니다. 또한 딥 러닝 및 DLAMI 사용에 익숙해야 합니다.

Conda 환경 활성화

다음 명령을 사용하여 MXNet-Neuron conda 환경을 활성화합니다.

```
source activate aws_neuron_mxnet_p36
```

현재 conda 환경을 종료하려면 다음을 실행합니다.

```
source deactivate
```

Resnet50 컴파일

다음 콘텐츠를 통해 **mxnet_compile_resnet50.py**라는 Python 스크립트를 생성합니다. 이 스크립트는 MXNet-Neuron 컴파일 Python API를 사용하여 ResNet-50 모델을 컴파일합니다.

```
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)
```

다음 명령을 사용하여 모델을 컴파일합니다.

```
python mxnet_compile_resnet50.py
```

컴파일이 끝날 때까지 몇 분 정도 소요될 수 있습니다. 컴파일이 완료되면 다음 파일이 현재 디렉터리에 저장됩니다.

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

ResNet50 추론

다음 콘텐츠를 통해 **mxnet_infer_resnet50.py**라는 Python 스크립트를 생성합니다. 이 스크립트는 샘플 이미지를 다운로드하고 이를 사용하여 컴파일된 모델에 대한 추론을 실행합니다.

```
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]))
```

다음 명령을 사용하여 컴파일된 모델에 대한 추론을 실행합니다.

```
python mxnet_infer_resnet50.py
```

출력은 다음과 같아야 합니다.

```
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
```

다음 단계

MXNet-Neuron 모델 제공 사용

MXNet-Neuron 모델 제공 사용

이 자습서에서는 사전 교육된 MXNet 모델을 사용하여 다중 모델 서버(MMS)로 실시간 이미지를 분류하는 방법을 알아봅니다. MMS는 기계 학습 또는 딥 러닝 프레임워크를 사용하여 교육 받은 딥 러닝 모델을 제공하기 위한 유연하고 사용하기 쉬운 도구입니다. 이 자습서에는 AWS Neuron을 사용하는 컴파일 단계와 MXNet을 사용하는 MMS 구현이 포함되어 있습니다.

Neuron SDK에 대한 자세한 내용은 AWS Neuron SDK 설명서를 참조하세요.

내용

- 사전 조건
- Conda 환경 활성화
- 예제 코드 다운로드
- 모델 컴파일

- 추론 실행

사전 조건

이 자습서를 사용하기 전에 AWS Neuron을 사용하여 DLAMI 인스턴스 시작의 설정 단계를 완료해야 합니다. 또한 딥 러닝 및 DLAMI 사용에 익숙해야 합니다.

Conda 환경 활성화

다음 명령을 사용하여 MXNet-Neuron conda 환경을 활성화합니다.

```
source activate aws_neuron_mxnet_p36
```

현재 conda 환경을 종료하려면 다음을 실행합니다.

```
source deactivate
```

예제 코드 다운로드

이 예제를 실행하려면 다음 명령을 사용하여 예제 코드를 다운로드합니다.

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

모델 컴파일

다음 콘텐츠를 통해 `multi-model-server-compile.py`라는 Python 스크립트를 생성합니다. 이 스크립트는 ResNet50 모델을 Inferentia 디바이스 대상으로 컴파일합니다.

```
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)
```

모델을 컴파일하려면 다음 명령을 사용합니다.

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

출력은 다음과 같아야 합니다.

```
...
[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!
```

다음 콘텐츠를 통해 `signature.json`이라는 파일을 생성하여 입력 이름과 셰이프를 구성합니다.

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

```
}
```

다음 명령을 사용하여 synset.txt 파일을 다운로드합니다. 이 파일은 ImageNet 예측 클래스의 이름 목록입니다.

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

model_server_template 폴더의 템플릿에 따라 사용자 지정 서비스 클래스를 생성합니다. 다음 명령을 사용하여 템플릿을 현재 작업 디렉터리에 복사합니다.

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

mxnet_model_service.py 모듈을 편집하여 mx.cpu() 컨텍스트를 다음과 같이 mx.neuron() 컨텍스트로 바꿉니다. 또한 MXNet-Neuron은 NDARray 및 Gluon API를 지원하지 않으므로 model_input에 불필요한 데이터 복사본을 주석 처리해야 합니다.

```
...
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]
```

다음 명령을 사용하여 model-archiver로 모델을 패키징합니다.

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

추론 실행

다중 모델 서버를 시작하고 다음 명령을 사용하여 RESTful API를 사용하는 모델을 로드합니다. neuron-rtd가 기본 설정으로 실행 중인지 확인합니다.

```
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
```

다음 명령을 사용하여 예제 이미지로 추론을 실행합니다.

```
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
```

출력은 다음과 같아야 합니다.

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

테스트 후 정리하려면 RESTful API를 통해 delete 명령을 실행하고 다음 명령을 사용하여 모델 서버를 중지합니다.

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

multi-model-server --stop
```

다음 결과가 표시됩니다.

```
{
  "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
```

PyTorch-Neuron 및 AWS Neuron 컴파일러 사용

PyTorch-Neuron 컴파일 API는 AWS Inferentia 디바이스에서 실행할 수 있는 모델 그래프를 컴파일하는 방법을 제공합니다.

훈련된 모델은 Inf1 인스턴스에서 배포하기 전에 Inferentia 대상에 컴파일되어야 합니다. 다음 자습서에서는 torchvision ResNet50 모델을 컴파일하고 저장된 TorchScript 모듈로 내보냅니다. 이러한 모델은 추론을 실행하는 데 사용됩니다.

편의상 이 자습서에서는 컴파일 및 추론 모두에 Inf1 인스턴스를 사용합니다. 실제로는 c5 인스턴스 패밀리와 같은 다른 인스턴스 유형을 사용하여 모델을 컴파일할 수 있습니다. 그런 다음 컴파일된 모델을 Inf1 추론 서버에 배포해야 합니다. 자세한 내용은 AWS Neuron PyTorch SDK 설명서를 참조하세요.

내용

사전 조건

이 자습서를 사용하기 전에 AWS Neuron을 사용하여 DLAMI 인스턴스 시작의 설정 단계를 완료해야 합니다. 또한 딥 러닝 및 DLAMI 사용에 익숙해야 합니다.

Conda 환경 활성화

다음 명령을 사용하여 PyTorch-Neuron conda 환경을 활성화합니다.

```
source activate aws_neuron_pytorch_p36
```

현재 conda 환경을 종료하려면 다음을 실행합니다.

```
source deactivate
```

Resnet50 컴파일

다음 콘텐츠를 통해 **pytorch_trace_resnet50.py**라는 Python 스크립트를 생성합니다. 이 스크립트는 PyTorch-Neuron 컴파일 Python API를 사용하여 ResNet-50 모델을 컴파일합니다.

> **Note**
>
> 토치비전 모델과 토치 패키지 버전 간에는 종속성이 있으며, 토치비전 모델을 컴파일할 때 이 점을 알고 있어야 합니다. 이러한 종속성 규칙은 pip를 통해 관리할 수 있습니다. Torchvision==0.6.1은 torch==1.5.1 릴리스와 일치하고, torchvision==0.8.2는 torch==1.7.1 릴리스와 일치합니다.

```
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")
```

컴파일 스크립트를 실행합니다.

```
python pytorch_trace_resnet50.py
```

컴파일이 끝날 때까지 몇 분 정도 소요될 수 있습니다. 컴파일이 완료되면 컴파일된 모델이 로컬 디렉터리에 `resnet50_neuron.pt`로 저장됩니다.

ResNet50 추론

다음 콘텐츠를 통해 **pytorch_infer_resnet50.py**라는 Python 스크립트를 생성합니다. 이 스크립트는 샘플 이미지를 다운로드하고 이를 사용하여 컴파일된 모델에 대한 추론을 실행합니다.

```
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) )
```

다음 명령을 사용하여 컴파일된 모델에 대한 추론을 실행합니다.

```
python pytorch_infer_resnet50.py
```

출력은 다음과 같아야 합니다.

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

ARM64 DLAMI

AWS ARM64 GPU DLAMIs는 딥 러닝 워크로드에 높은 성능과 비용 효율성을 제공하도록 설계되었습니다. 특히 G5g 인스턴스 유형에는 Arm64-based AWS Graviton2 프로세서가 탑재되어 있습니다. 이 프로세서는 처음부터 클라우드에서 워크로드를 실행하는 방식에 맞게 최적화 AWS 되어 있습니다. AWS ARM64 GPU DLAMIs는 Docker, NVIDIA Docker, NVIDIA 드라이버, CUDA, CuDNN, NCCL뿐만 아니라 TensorFlow 및 PyTorch와 같은 인기 있는 기계 학습 프레임워크로 사전 구성되어 있습니다.

G5g 인스턴스 유형을 사용하면 Graviton2의 가격 및 성능 이점을 활용하여 GPU 가속을 지원하는 x86 기반 인스턴스보다 훨씬 저렴한 비용으로 GPU 가속 딥 러닝 모델을 배포할 수 있습니다.

ARM64 DLAMI 선택

선택한 ARM64 DLAMI로 G5g 인스턴스를 시작합니다.

DLAMI 시작에 대한 단계별 지침은 DLAMI 시작 및 구성을 참조하세요.

최신 ARM64 DLAMI 목록은 DLAMI 릴리스 노트를 참조하세요.

시작하기

다음 섹션에서는 ARM64 DLAMI 사용을 시작하는 방법을 보여 줍니다.

내용

- ARM64 GPU PyTorch DLAMI 사용

ARM64 GPU PyTorch DLAMI 사용

AWS Deep Learning AMIs 는 Arm64 프로세서 기반 GPUs와 함께 사용할 준비가 되었으며 PyTorch에 최적화되어 제공됩니다. ARM64 GPU PyTorch DLAMI에는 딥 러닝 훈련 및 추론 사용 사례를 위해 PyTorch, TorchVision 및 TorchServe로 사전 구성된 Python 환경이 포함되어 있습니다.

내용

- PyTorch Python 환경 확인
- PyTorch를 사용하여 교육 샘플 실행
- PyTorch로 추론 샘플 실행

PyTorch Python 환경 확인

G5g 인스턴스에 연결하고 다음 명령을 사용하여 기본 Conda 환경을 활성화합니다.

```
source activate base
```

명령 프롬프트는 PyTorch, TorchVision 및 기타 라이브러리가 포함된 기본 Conda 환경에서 작업하고 있음을 나타내야 합니다.

```
(base) $
```

PyTorch 환경의 기본 도구 경로를 확인합니다.

```
(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)
```

PyTorch를 사용하여 교육 샘플 실행

샘플 MNIST 교육 작업을 실행합니다.

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

출력은 다음과 비슷한 형태가 됩니다.

```
...
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%)
```

PyTorch로 추론 샘플 실행

다음 명령을 사용하여 사전 학습된 densenet161 모델을 다운로드하고 TorchServe를 사용하여 추론을 실행하세요.

```
# 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
```

출력은 다음과 비슷한 형태가 됩니다.

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

다음 명령을 사용하여 densenet161 모델을 등록 취소하고 서버를 중지하세요.

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

출력은 다음과 비슷한 형태가 됩니다.

```
{
```

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

Inference

이 섹션은 DLAMI의 프레임워크 및 도구를 사용하여 추론 실행 방법에 대한 자습서입니다

추론 도구

- TensorFlow Serving

모델 제공

다음은 Conda를 사용하는 DLAMI에 설치된 모델 제공 옵션입니다. 옵션 중 한 가지를 클릭하여 사용법을 알아보십시오.

주제

- TensorFlow Serving
- TorchServe

TensorFlow Serving

TensorFlow Serving은 머신 러닝 모델을 위한 유연한 고성능 서비스 시스템입니다.

`tensorflow-serving-api` 에는 단일 프레임워크 DLAMI가 사전 설치되어 있습니다. 텐서플로우 서빙을 사용하려면 먼저 TensorFlow 환경을 활성화합니다.

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

그런 다음 원하는 텍스트 편집기를 사용하여 다음 내용이 들어 있는 스크립트를 생성합니다. 이름을 `test_train_mnist.py`로 지정합니다. 이 스크립트는 이미지를 분류하는 신경망 기계 학습 모델을 훈련하고 평가하는 TensorFlow 자습서에서 참조됩니다.

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

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

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

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

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

이제 서버 위치, 포트 및 허스키 사진의 파일 이름을 파라미터로 전달하는 스크립트를 실행합니다.

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

이 스크립트에서 결과가 출력되려면 시간이 걸리므로 인내심을 가지고 기다리십시오. 훈련이 완료되면 다음이 표시됩니다.

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

더 많은 기능과 예제

TensorFlow Serving에 대한 자세한 내용은 TensorFlow 웹 사이트를 참조하세요.

TorchServe

TorchServe는 PyTorch에서 내보낸 딥 러닝 모델을 제공하기 위한 유연한 도구입니다. TorchServe에는 Conda를 사용하는 Deep Learning AMI가 사전 설치되어 있습니다.

TorchServe 사용에 대한 자세한 내용은 PyTorch용 모델 서버 설명서를 참조하세요.

주제

TorchServe에서 이미지 분류 모델 서비스

이 자습서에서는 TorchServe로 이미지 분류 모델을 서비스하는 방법을 보여줍니다. PyTorch에서 제공하는 DenseNet-161 모델을 사용합니다. 서버가 실행되면 예측 요청을 수신합니다. 예를 들어 고양이의 이미지와 같은 이미지를 업로드할 때 서버는 모델이 교육한 클래스 가운데 가장 일치하는 5개 클래스에 대한 예측을 반환합니다.

TorchServe에서 예제 이미지 분류 모델을 서비스하려면

1. Conda를 사용하는 DLAMI(버전 34 이상)의 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에 연결합니다.
2. `pytorch_p310` 환경을 활성화합니다.

   ```
   source activate pytorch_p310
   ```

3. TorchServe 리포지토리를 복제한 다음 모델을 저장할 디렉토리를 생성합니다.

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

4. 모델 아카이버를 사용하여 모델을 보관합니다. `extra-files` 매개변수는 `TorchServe` 리포지토리의 파일을 사용하므로 필요한 경우 경로를 업데이트하세요. 모델 아카이버에 대한 자세한 내용은 TorchServe용 Torch 모델 아카이버를 참조하세요.

   ```
   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. TorchServe를 실행하여 엔드포인트를 시작합니다. > /dev/null을 추가하면 로그 출력이 중지됩니다.

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

6. 고양이의 이미지를 다운로드한 다음 TorchServe 예측 엔드포인트로 전송합니다.

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

예측 엔드포인트는 다음 상위 5개의 예측과 유사한 예측을 JSON으로 반환합니다. 여기에서 이미지는 47%의 확률로 이집트 고양이를 포함하고, 그 다음으로 46%의 확률로 태비 고양이를 포함합니다.

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

7. 테스트를 마친 후 서버를 중단합니다.

```
torchserve --stop
```

기타 예제

TorchServe에는 사용자가 DLAMI 인스턴스에서 실행할 수 있는 다양한 예제가 있습니다. 이러한 예제는 TorchServe 프로젝트 리포지토리에서 볼 수 있습니다.

추가 정보

Docker와 최신 TorchServe 기능을 사용하여 TorchServe를 설정하는 방법을 포함한 더 많은 TorchServe 설명서는 GitHub의 TorchServe 프로젝트 페이지를 참조하세요.

DLAMI 업그레이드

이 섹션에서 DLAMI 업그레이드에 관한 정보와 DLAMI에서의 소프트웨어 업데이트에 대한 팁을 확인하세요.

패치 및 업데이트가 발표되는 즉시 적용하여 운영 체제와 기타 설치된 소프트웨어를 항상 최신 상태로 유지하십시오.

Amazon Linux 또는 Ubuntu 사용자가 DLAMI에 로그인할 때 사용 가능한 업데이트가 있으면 알림 메시지와 함께 업데이트를 위한 지침이 표시됩니다. Amazon Linux 유지 관리에 대한 자세한 내용은 인스턴스 소프트웨어 업데이트 섹션을 참조하세요. Ubuntu 인스턴스는 공식 Ubuntu 설명서를 참조하십시오.

Windows에서는 Windows 업데이트에서 소프트웨어 및 보안 업데이트를 정기적으로 점검하십시오. 원한다면 업데이트가 자동으로 적용되게 할 수 있습니다.

> **Important**
>
> Meltdown 및 Spectre 취약성과 이를 해결하기 위해 운영 체제를 패치하는 방법에 대한 자세한 내용은 Security Bulletin AWS-2018-013을 참조하세요.

주제

최신 버전으로 DLAMI 업그레이드

DLAMI의 시스템 이미지는 새로운 딥 러닝 프레임워크 릴리스, CUDA 및 기타 소프트웨어 업데이트, 그리고 성능 튜닝을 이용하기 위해 정기적으로 업데이트됩니다. DLAMI를 일정 기간 사용하여 업데이트를 원할 경우 새 인스턴스를 시작해야 합니다. 또한 데이터 세트, 체크포인트 등 귀중한 정보를 모두 수동으로 이전해야 합니다. 대신, Amazon EBS를 사용하여 데이터를 유지하고 새 DLAMI에 연결할 수 있습니다. 이 방식을 사용하면 자주 업그레이드하면서도 데이터 이전에 소요되는 시간을 최소화할 수 있습니다.

> **Note**
>
> Amazon EBS 볼륨을 DLAMI 간에 이동하고 연결할 때 두 DLAMI와 새 볼륨이 동일한 가용 영역에 위치해야 합니다.

1. Amazon EC2 콘솔을 사용하여 새 Amazon EBS 볼륨을 생성합니다. 자세한 지침은 Amazon EBS 볼륨 생성을 참조하세요.
2. 새로 생성한 Amazon EBS 볼륨을 기존 DLAMI에 연결합니다. 자세한 지침은 Amazon EBS 볼륨 연결을 참조하세요.
3. 데이터(예: 데이터 세트, 체크포인트, 구성 파일)를 이전합니다.
4. DLAMI를 실행합니다. 자세한 지침은 DLAMI 인스턴스 설정 섹션을 참조하세요.
5. 기존 DLAMI에서 아마존 EBS 볼륨을 분리합니다. 자세한 지침은 Amazon EBS 볼륨 분리를 참조하세요.
6. 새 DLAMI에 Amazon EBS 볼륨을 연결합니다. 2단계의 지침에 따라 볼륨을 연결합니다.
7. 새 DLAMI에서 데이터를 사용 가능한지 확인한 후 기존 DLAMI를 중지 및 종료합니다. 정리에 대한 자세한 지침은 DLAMI 인스턴스 정리 섹션을 참조하세요.

소프트웨어 업데이트에 관한 팁

때때로 DLAMI에서 소프트웨어를 수동으로 업데이트해야 하는 경우가 있을 수 있습니다. 일반적으로 Python 패키지 업데이트에는 `pip`을 사용하는 것이 좋습니다. 또한 Conda를 사용하는 DLAMI의 Conda 환경에서 패키지를 업데이트할 때는 `pip`을 사용해야 합니다. 지침 업그레이드 및 설치를 위해서는 특정 프레임워크 또는 소프트웨어의 웹 사이트를 참조하십시오.

> **Note**
>
> 패키지 업데이트의 성공을 보장할 수 없습니다. 호환되지 않는 종속성이 있는 환경에서 패키지를 업데이트하려고 하면 실패할 수 있습니다. 이러한 경우 라이브러리 관리자에게 문의하여 패키지 종속성을 업데이트할 수 있는지 확인해야 합니다. 또는 업데이트를 허용하는 방식으로 환경을 수정해 볼 수도 있습니다. 그러나 이러한 수정은 기존 패키지를 제거하거나 업데이트하는 것을 의미할 수 있으며, 이는 더 이상 이 환경의 안정성을 보장할 수 없음을 의미합니다.

AWS Deep Learning AMIs 에는 많은 Conda 환경과 많은 패키지가 사전 설치되어 있습니다. 사전 설치된 패키지 수가 많기 때문에 호환성이 보장되는 패키지 세트를 찾기가 어렵습니다. “환경이 일치하지

않습니다. 패키지 플랜이 올바른지 확인하세요" 경고가 표시될 수 있습니다. DLAMI는 DLAMI에서 제공하는 모든 환경이 올바른지 확인하지만 사용자가 설치한 패키지가 제대로 작동한다고 보장할 수는 없습니다.

새 업데이트 알림 받기

> **Note**
>
> AWS Deep Learning AMIs에는 보안 패치에 대한 주간 릴리스 주기가 있습니다. 이러한 증분 보안 패치에 대한 릴리스 알림은 공식 릴리스 노트에 포함되어 있지 않을 수 있지만 전송됩니다.

새 DLAMI가 릴리스될 때마다 알림을 받을 수 있습니다. 다음 주제에 대한 알림이 Amazon SNS에 게시됩니다.

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

새 DLAMI가 게시되면 메시지가 여기에 게시됩니다. AMI의 버전, 메타데이터 및 리전 AMI ID가 메시지에 포함됩니다.

이러한 메시지는 여러 방식으로 수신할 수 있습니다. 다음 방법을 사용하는 것이 좋습니다.

1. Amazon SNS 콘솔을 엽니다.
2. 필요한 경우 탐색 모음에서 AWS 리전을 미국 서부(오레곤)로 변경합니다. 구독 중인 SNS 알림이 생성된 리전을 선택해야 합니다.
3. 탐색 창에서 구독, 구독 생성을 선택합니다.
4. 구독 생성 대화 상자에서 다음과 같이 수행합니다.
   a. 주제 ARN의 경우, 다음 Amazon 리소스 이름(ARN)을 복사합니다. **`arn:aws:sns:us-west-2:767397762724:dlami-updates`**
   b. 프로토콜에서 [Amazon SQS, AWS Lamda, Email, Email-JSON] 중 하나를 선택합니다.
   c. 엔드포인트에 알림을 받는 데 사용할 리소스의 이메일 주소 또는 Amazon 리소스 이름(ARN)을 입력합니다.
   d. 구독 생성을 선택합니다.
5. AWS 알림 - 구독 확인이라는 제목의 확인 이메일을 받게 됩니다. 이메일을 열고 구독 확인을 선택하여 구독 신청을 완료합니다.

의 보안 AWS Deep Learning AMIs

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS AWS 서비스 에서 실행되는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 AWS 제공 범위 내 서비스규정 준수 프로그램 제공 범위 내 서비스를 AWS Deep Learning AMIs참조하세요.
- 클라우드의 보안 - 사용자의 책임은 AWS 서비스 사용하는에 따라 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 DLAMI 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 섹션에서는 보안 및 규정 준수 목표를 충족하도록 DLAMI를 구성하는 방법을 보여줍니다. 또한 DLAMI 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 다른를 사용하는 방법을 알아봅니다.

자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2의 보안을 참조하세요.

주제

의 데이터 보호 AWS Deep Learning AMIs

AWS 공동 책임 모델의 데이터 보호에 적용됩니다 AWS Deep Learning AMIs. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대

한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션과 내부의 모든 기본 보안 제어를 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 DLAMI 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

에 대한 자격 증명 및 액세스 관리 AWS Deep Learning AMIs

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와주는 입니다. IAM 관리자는 DLAMI 리소스 사용에 대한 인증(로그인됨) 및 권한 부여(권한 있음) 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

자격 증명 및 액세스 관리에 대한 자세한 내용은 Amazon EC2의 자격 증명 및 액세스 관리를 참조하세요.

주제

ID를 통한 인증

인증은 자격 증명 AWS 으로에 로그인하는 방법입니다. , AWS 계정 루트 사용자 IAM 사용자 또는 IAM 역할을 수임하여 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어는 멀티 팩터 인증(MFA)을 사용하여 계정의 보안을 강화할 것을 AWS 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 작업을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 AWS Management Console수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 수 있습니다. 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니

다. 그러나 일부 에서는 (역할을 프록시로 사용하는 대신) 정책을 리소스에 직접 연결할 AWS 서비스 수 있습니다. 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.

- 교차 서비스 액세스 - 일부는 다른의 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하려면 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신하는 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결 AWS 될 때 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거

나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 기업이 소유한 여러 AWS 계정 을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속해 있는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명의 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon EMR을 사용하는 IAM

Amazon EMR에서 IAM을 사용하여 사용자, AWS 리소스, 그룹, 역할 및 정책을 정의할 수 있습니다. AWS 서비스 이러한 사용자와 역할이 액세스할 수 있는 권한을 제어할 수도 있습니다.

Amazon EMR과 IAM 사용에 대한 자세한 내용은 Amazon EMR에AWS Identity and Access Management 사용을 참조하세요.

에 대한 규정 준수 검증 AWS Deep Learning AMIs

타사 감사자는 여러 규정 준수 프로그램의 AWS Deep Learning AMIs 일환으로의 보안 및 AWS 규정 준수를 평가합니다. 지원되는 규정 준수 프로그램에 대한 자세한 내용은 Amazon EC2의 규정 준수 확인을 참조하세요.

특정 규정 준수 프로그램의 범위 AWS 서비스 내 목록은 규정 준수 프로그램 AWS 제공 범위 내 서비스 규정 준수 프로그램 . 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하세요.

DLAMI 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서: 이 배포 안내서에서는 아키텍처 고려 사항에 관해 설명하고 AWS에서 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- AWS Config 개발자 안내서의 AWS Config 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub - 이를 AWS 서비스 통해 내 보안 상태를 포괄적으로 볼 수 있습니다 AWS. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다.

의 복원력 AWS Deep Learning AMIs

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.는 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전 제공합니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

데이터 복원성 및 백업 요구 사항을 지원하는 Amazon EC2 기능에 대한 자세한 정보는 Amazon EC2 사용 설명서의 Amazon EC2의 복원성을 참조하세요.

의 인프라 보안 AWS Deep Learning AMIs

의 인프라 보안 AWS Deep Learning AMIs 은 Amazon EC2에서 지원합니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2의 인프라 보안을 참조하세요.

AWS Deep Learning AMIs 인스턴스 모니터링

모니터링은 AWS Deep Learning AMIs 인스턴스 및 기타 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. DLAMI 인스턴스에는 Amazon CloudWatch에 GPU 사용 통계를 보고하는 유틸리티를 포함한 여러 가지 GPU 모니터링 도구가 제공됩니다. 자세한 내용은 GPU 모니터링 및 최적화 및 Amazon EC2 사용 설명서의 Amazon EC2 리소스 모니터링을 참조하세요.

DLAMI 인스턴스에 대한 사용량 추적 거부

다음 AWS Deep Learning AMIs 운영 체제 배포에는가 인스턴스 유형, 인스턴스 ID, DLAMI 유형 및 OS 정보를 수집할 AWS 수 있는 코드가 포함됩니다.

> **Note**
>
> AWS 는 DLAMI 내에서 사용하는 명령과 같은 DLAMI에 대한 다른 정보를 수집하거나 유지하지 않습니다.

- Amazon Linux 2
- Amazon Linux 2023

- Ubuntu 20.04
- Ubuntu 22.04

사용량 추적을 거부하려면

원할 경우 새 DLAMI 인스턴스에 대한 사용량 추적을 거부할 수 있습니다. 사용량 추적을 거부하려면 시작 중 Amazon EC2 인스턴스에 태그를 추가해야 합니다. 태그는 관련 값이 `OPT_OUT_TRACKING`으로 설정된 `true` 키를 사용해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 리소스에 태그 지정을 참조하세요.

DLAMI 지원 정책

여기에서 AWS Deep Learning AMIs (DLAMI)에 대한 지원 정책의 세부 정보를 찾을 수 있습니다.

AWS 현재가 지원하는 DLAMI 프레임워크 및 운영 체제 목록은 DLAMI 지원 정책 페이지를 참조하세요. 다음 용어는 지원 정책 페이지 및이 페이지에 언급된 모든 DLAMIs에 적용됩니다.

- 현재 버전은 프레임워크 버전을 x.y.z 형식으로 지정합니다. 프레임워크 버전 x는 메이저 버전, y는 마이너 버전, z는 패치 버전을 의미합니다. 예를 들어 TensorFlow 2.10.1이라면 메이저 버전이 2, 마이너 버전이 10, 패치 버전은 1입니다.
- 패치 종료는가 특정 프레임워크 또는 운영 체제 버전을 AWS 지원하는 기간을 지정합니다.

특정 DLAMI에 대한 자세한 내용은 DLAMI 릴리스 노트 섹션을 참조하세요.

DLAMI 지원 FAQs

- 어떤 프레임워크 버전에 보안 패치가 적용되나요?
- 보안 패치를 받는 운영 체제는 무엇입니까?
- 새 프레임워크 버전이 릴리스될 때 AWS 게시되는 이미지는 무엇입니까?
- 새로운 SageMaker AI/AWS 기능을 가져오는 이미지는 무엇입니까?
- 지원되는 프레임워크 표에서 현재 버전을 어떻게 확인하나요?
- 지원되는 테이블에 없는 버전을 실행하는 경우 어떻게 해야 합니까?
- DLAMIs 프레임워크 버전의 이전 패치 버전을 지원하나요?
- 지원되는 프레임워크 버전에 대해 최신 패치가 적용된 이미지를 찾으려면 어떻게 해야 하나요?
- 새 이미지는 얼마나 자주 릴리스되나요?
- 워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요?
- 패치가 적용되거나 업데이트된 새 프레임워크 버전을 사용할 수 있게 되면 어떻게 해야 하나요?
- 프레임워크 버전을 변경하지 않고도 종속성이 업데이트되나요?
- 내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?
- 더 이상 활발하게 유지 관리되지 않는 프레임워크 버전의 이미지도 패치가 적용되나요?
- 이전 프레임워크 버전을 사용하고 싶습니다.

- 프레임워크 및 해당 버전의 지원 변경 사항을 최신 상태로 유지하고 싶습니다.
- Anaconda 리포지토리를 사용하려면 상용 라이선스가 필요한가요?

어떤 프레임워크 버전에 보안 패치가 적용되나요?

프레임워크 버전이 지원 AWS Deep Learning AMIs 정책 테이블의 지원되는 프레임워크 버전 아래에 있는 경우 보안 패치를 가져옵니다.

보안 패치를 받는 운영 체제는 무엇입니까?

운영 체제가 지원 AWS Deep Learning AMIs 정책 테이블의 지원되는 운영 체제 버전 아래에 나열되면 보안 패치가 표시됩니다.

새 프레임워크 버전이 릴리스될 때 AWS 게시되는 이미지는 무엇입니까?

TensorFlow와 PyTorch의 새 버전이 출시된 직후 새로운 DLAMI를 게시합니다. 여기에는 프레임워크의 메이저 버전, 메이저 마이너 버전, 메이저 마이너 패치 버전이 포함됩니다. 또한 새 버전의 드라이버와 라이브러리가 출시되면 이미지도 업데이트됩니다. 이미지 유지 관리에 대한 자세한 내용은 내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요? 섹션을 참조하세요.

새로운 SageMaker AI/AWS 기능을 가져오는 이미지는 무엇입니까?

새로운 기능은 일반적으로 PyTorch와 TensorFlow용 DLAMIS의 최신 버전에서 릴리스됩니다. 새로운 SageMaker AI 또는 AWS 기능에 대한 자세한 내용은 특정 이미지의 릴리스 정보를 참조하세요. 사용 가능한 DLAMI 목록은 DLAMI 릴리스 노트를 참조하세요. 이미지 유지 관리에 대한 자세한 내용은 내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요? 섹션을 참조하세요.

지원되는 프레임워크 표에서 현재 버전을 어떻게 확인하나요?

AWS Deep Learning AMIs 지원 정책 테이블의 현재 버전은가 GitHub에서 AWS 사용할 수 있는 최신 프레임워크 버전을 나타냅니다. 각 최신 릴리스에는 DLAMI의 드라이버, 라이브러리 및 관련 패키지에 대한 업데이트가 포함됩니다. 이미지 유지 관리에 대한 자세한 내용은 내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요? 섹션을 참조하세요.

지원되는 테이블에 없는 버전을 실행하는 경우 어떻게 해야 합니까?

AWS Deep Learning AMIs 지원 정책 테이블에 없는 버전을 실행하는 경우 최신 드라이버, 라이브러리 및 관련 패키지가 없을 수 있습니다. up-to-date 버전의 경우 선택한 최신 DLAMI를 사용하여 지원되

는 프레임워크 또는 운영 체제 중 하나로 업그레이드하는 것이 좋습니다. 사용 가능한 DLAMI 목록은 DLAMI 릴리스 노트를 참조하세요.

DLAMIs 프레임워크 버전의 이전 패치 버전을 지원하나요?

아니요. 지원 정책 표에 명시된 대로 초기 GitHub 릴리스로부터 365일 후에 릴리스된 각 프레임워크의 최신 메이저 버전의 최신 패치 버전을 지원합니다. AWS Deep Learning AMIs 자세한 내용은 지원되는 테이블에 없는 버전을 실행하는 경우 어떻게 해야 합니까? 섹션을 참조하세요.

지원되는 프레임워크 버전에 대해 최신 패치가 적용된 이미지를 찾으려면 어떻게 해야 하나요?

최신 프레임워크 버전에서 DLAMI를 사용하려면 AWS CLI 또는 SSM 파라미터를 사용하여 DLAMI ID를 검색하고 이를 사용하여 EC2 콘솔을 사용하여 DLAMI를 시작할 수 있습니다. AWS Deep Learning AMIs ID를 검색하는 샘플 AWS CLI 또는 SSM 파라미터 명령은 DLAMI 릴리스 정보 페이지 단일 프레임워크 DLAMI 릴리스 정보를 참조하세요. 선택한 프레임워크 버전은 AWS Deep Learning AMIs 지원 정책 테이블의 지원되는 프레임워크 버전 아래에 나열되어야 합니다.

새 이미지는 얼마나 자주 릴리스되나요?

업데이트된 패치 버전을 제공하는 것이 저희의 최우선 과제입니다. 따라서 패치가 적용된 이미지를 가능한 한 빨리 정기적으로 제작합니다. 새로 패치된 프레임워크 버전(예: TensorFlow 2.9 ~ TensorFlow 2.9.1)과 새로운 마이너 릴리스 버전(예: TensorFlow 2.9 ~ TensorFlow 2.10)을 모니터링하여 가능한 빠른 시일 내에 사용할 수 있게 최선을 다하고 있습니다. 기존 버전의 TensorFlow가 CUDA 새 버전과 함께 출시되면 CUDA 새 버전을 지원하는 TensorFlow 버전용 새 DLAMI가 릴리스됩니다.

워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요?

아니요. DLAMI에 대한 패치 업데이트는 “실시간” 업데이트가 아닙니다.

새 EC2 인스턴스를 켜고 워크로드와 스크립트를 마이그레이션한 다음 이전 인스턴스를 꺼야 합니다.

패치가 적용되거나 업데이트된 새 프레임워크 버전을 사용할 수 있게 되면 어떻게 해야 하나요?

DLAMI의 변경 사항에 대한 알림을 받으려면 관련 DLAMI에 대한 알림을 구독하십시오. 새 업데이트에 대한 알림 수신을 참조하세요.

프레임워크 버전을 변경하지 않고도 종속성이 업데이트되나요?

종속성은 프레임워크 버전을 변경하지 않고 업데이트됩니다. 하지만 종속성 업데이트로 인해 비호환성이 발생하는 경우 다른 버전으로 이미지를 생성합니다. 업데이트된 종속성 정보는 DLAMI용 릴리스 노트를 확인하세요.

내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?

DLAMI 이미지는 변경이 불가능합니다. 생성된 후에는 변경되지 않습니다. 프레임워크 버전에 대한 능동 지원이 종료되는 이유는 크게 네 가지입니다.

- 프레임워크 버전(패치) 업그레이드
- AWS 보안 패치
- 패치 종료일(에이징 아웃)
- 종속성 지원 종료

> **Note**
>
> 버전 패치 업그레이드 및 보안 패치의 빈도가 높으므로 DLAMI의 릴리스 노트 페이지를 자주 확인하고 변경이 있을 때 업그레이드하는 것이 좋습니다.

프레임워크 버전(패치) 업그레이드

TensorFlow 2.7.0을 기반으로 하는 DLAMI 워크로드가 있고 TensorFlow가 GitHub에서 버전 2.7.1을 릴리스한 경우는 TensorFlow 2.7.1을 사용하여 새 DLAMI를 AWS 릴리스합니다. TensorFlow 2.7.1이 포함된 새 이미지가 출시되면 2.7.0의 이전 이미지는 더 이상 활발하게 유지되지 않습니다. TensorFlow 2.7.0을 포함하는 DLAMI는 추가 패치를 받지 않습니다. 그러면 TensorFlow 2.7의 DLAMI 릴리스 노트 페이지가 최신 정보로 업데이트됩니다. 각 마이너 패치에 대한 개별 릴리스 노트 페이지는 없습니다.

패치 업그레이드로 인해 생성된 새 DLAMI는 새 AMI ID로 지정됩니다.

AWS 보안 패치

TensorFlow 2.7.0을 사용하는 이미지를 기반으로 하는 워크로드가 있고 보안 패치를 AWS 적용하면 TensorFlow 2.7.0에 대한 DLAMI의 새 버전이 릴리스됩니다. TensorFlow 2.7.0이 설치된 이전 버전의

이미지는 더 이상 활발하게 유지 관리되지 않습니다. 자세한 워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요? 섹션을 참조하세요. 최신 DLAMI 검색 방법은 지원되는 프레임워크 버전에 대해 최신 패치가 적용된 이미지를 찾으려면 어떻게 해야 하나요? 섹션을 참조하세요.

패치 업그레이드로 인해 생성된 새 DLAMI는 새 AMI ID로 지정됩니다.

패치 종료일(에이징 아웃)

DLAMI는 GitHub 출시일로부터 365일 후에 패치 날짜가 종료됩니다.

다중 프레임워크 DLAMI의 경우 프레임워크 버전 중 하나가 업데이트되면 업데이트된 버전의 새 DLAMI가 필요합니다. 이전 프레임워크 버전의 DLAMI는 더 이상 활발하게 유지 관리되지 않습니다.

> **Important**
>
> 주요 프레임워크 업데이트가 있는 경우에는 예외가 적용됩니다. 예를 들어, TensorFlow 1.15가 TensorFlow 2.0으로 업데이트되면 GitHub 릴리스일로부터 2년 또는 원본 프레임워크 유지 관리팀이 지원을 중단한 날로부터 6개월 간(둘 중 더 빠른 날짜) 최신 버전의 TensorFlow 1.15를 계속 지원합니다.

종속성 지원 종료

Python 3.6이 설치된 TensorFlow 2.7.0 DLAMI 이미지에서 워크로드를 실행하고 있고 해당 버전의 Python이 지원 종료로 표시된 경우 Python 3.6을 기반으로 하는 모든 DLAMI 이미지는 더 이상 활발하게 유지 관리되지 않습니다. 마찬가지로 Ubuntu 16.04와 같은 OS 버전이 지원 종료로 표시되면 Ubuntu 16.04에 종속된 모든 DLAMI 이미지는 더 이상 활발하게 유지 관리되지 않습니다.

더 이상 활발하게 유지 관리되지 않는 프레임워크 버전의 이미지도 패치가 적용되나요?

아니요. 더 이상 활발하게 관리되지 않는 이미지는 새 릴리스로 제공되지 않습니다.

이전 프레임워크 버전을 사용하고 싶습니다.

이전 프레임워크 버전에서 DLAMI를 사용하려면 DLAMI ID를 검색하고 이를 사용하여 EC2 콘솔을 사용하여 DLAMI를 시작하세요. AMI ID를 검색하는 AWS CLI 명령의 경우 단일 프레임워크 DLAMI 릴리스 정보의 릴리스 정보 페이지를 참조하세요.

프레임워크 및 해당 버전의 지원 변경 사항을 최신 상태로 유지하고 싶습니다.

AWS Deep Learning AMIs 프레임워크 지원 정책 표, DLAMI 릴리스 노트를 사용하여 DLAMI 프레임워크 및 버전에 대한 최신 정보를 확인하세요.

Anaconda 리포지토리를 사용하려면 상용 라이선스가 필요한가요?

Anaconda는 특정 사용자를 위한 상용 라이선스 모델로 전환했습니다. 적극적으로 유지 관리되는 DLAMi는 Anaconda 채널에서 공개적으로 사용 가능한 오픈 소스 버전의 Conda(conda-forge) 로 마이그레이션되었습니다.

DLAMI에 대한 중요 NVIDIA 드라이버 변경 사항

2023년 11월 15일, AWS 는 DLAMI가 사용하는 NIVIDA 드라이버와 관련하여 AWS Deep Learning AMIs (DLAMI)를 변경했습니다. DLAMIs 변경 사항과 DLAMI 사용에 영향을 미치는지 여부에 대한 자세한 내용은 DLAMI NVIDIA 드라이버 변경 사항 FAQ 섹션을 참조하세요.

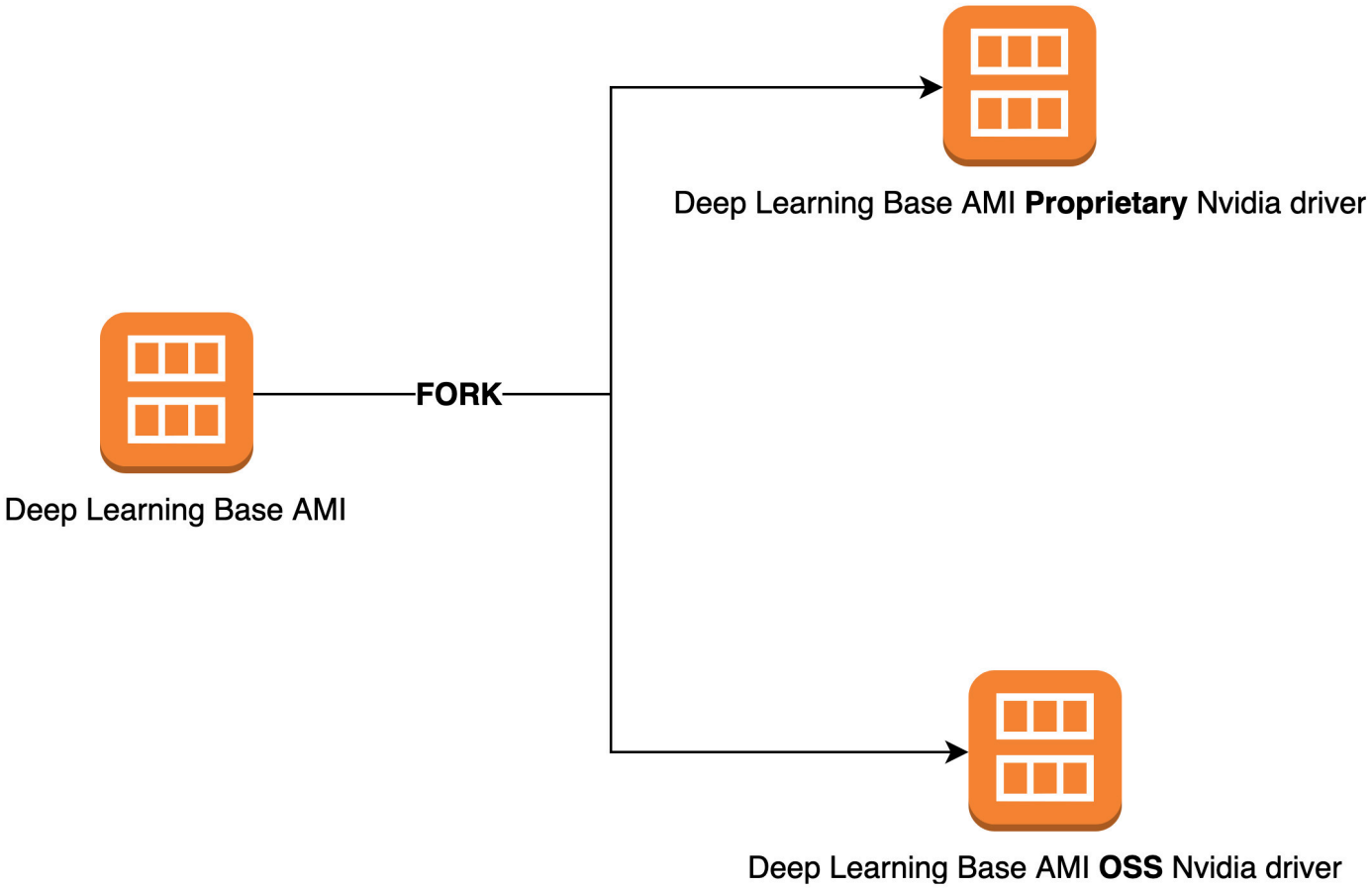
DLAMI NVIDIA 드라이버 변경 사항 FAQ

무엇이 변경되었나요?

DLAMI를 다음 두 개별 그룹으로 나눴습니다.

- NVIDIA 독점 드라이버(P3, P3dn, G3 지원)를 사용하는 DLAMI
- NVIDIA OSS 드라이버(G4dn, G5, P4, P5 지원)를 사용하는 DLAMI

따라서 두 범주 각각에 대해 새 이름과 새 AMI ID를 갖는 새 DLAMI를 생성했습니다. 이러한 DLAMI는 서로 바꿔 사용할 수 없습니다. 즉, 한 그룹의 DLAMI는 다른 그룹이 지원하는 인스턴스를 지원하지 않습니다. 예를 들어 P5를 지원하는 DLAMI는 G3를 지원하지 않으며 G3를 지원하는 DLAMI는 P5를 지원하지 않습니다.



이 변경이 필요한 이유는 무엇인가요?

이전에는 NVIDIA GPU용 DLAMI에 NVIDIA의 독점 커널 드라이버가 포함되었습니다. 그런데 업스트림 Linux 커널 커뮤니티는 NVIDIA GPU 드라이버와 같은 독점 커널 드라이버가 다른 커널 드라이버와 통신하지 못하도록 격리하는 변경을 수락했습니다. 이 변경으로 인해 GPU가 분산 훈련에 EFA를 효율적으로 사용할 수 있도록 하는 메커니즘인 GPUDirect RDMA가 P4 및 P5 시리즈 인스턴스에서 비활성화됩니다. 그 결과, DLAMI는 이제 오픈 소스 EFA 드라이버에 연결된 OpenRM 드라이버(NVIDIA 오픈 소스 드라이버)를 사용하여 G4dn, G5, P4 및 P5를 지원합니다. 그러나 이 OpenRM 드라이버는 이전 인스턴스(예: P3 및 G3)를 지원하지 않습니다. 따라서 두 인스턴스 유형을 모두 지원하는 안전한 최신 고성능 DLAMI를 계속 제공하기 위해 DLAMI를 두 그룹으로 나눴습니다. 한 그룹은 OpenRM 드라이버(G4dn, G5, P4, P5 지원)를 사용하고 다른 하나는 이전 독점 드라이버(P3, P3dn, G3 지원)를 사용합니다.

이 변경 사항은 어느 DLAMI에 영향을 미쳤나요?

이 변경 사항은 모든 DLAMI에 영향을 미쳤습니다.

이것이 사용자에게 의미하는 바는 무엇인가요?

지원되는 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스 유형에서 실행되는 한 모든 DLAMI는 기능, 성능 및 보안을 계속 제공합니다. DLAMI가 지원하는 EC2 인스턴스 유형을 확인하려면 해당 DLAMI의 릴리스 노트를 확인한 다음 지원되는 EC2 인스턴스를 찾습니다. 현재 지원되는 DLAMI 옵션 목록 및 릴리스 노트 링크는 DLAMI 릴리스 노트 섹션을 참조하세요.

또한 현재 DLAMIs를 호출하려면 correct AWS Command Line Interface (AWS CLI) 명령을 사용해야 합니다.

P3, P3dn 및 G3를 지원하는 Base DLAMI의 경우 다음 명령을 사용합니다.

```
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
```

G4dn, G5, P4 및 P5를 지원하는 Base DLAMI의 경우 다음 명령을 사용합니다.

```
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
```

최신 DLAMI에서 기능 손실이 있나요?

아니요. 기능 손실은 없습니다. 최신 DLAMI는 지원되는 EC2 인스턴스 유형에서 실행되는 한 이전 DLAMI의 모든 기능, 성능 및 보안을 제공합니다.

이 변경 사항이 Deep Learning Containers에 영향을 미쳤나요?

아니요.이 변경 사항은 NVIDIA 드라이버를 포함하지 않기 때문에 AWS Deep Learning Containers에 영향을 주지 않았습니다. 하지만 기본 인스턴스와 호환되는 AMI에서 Deep Learning Containers를 실행해야 합니다.

DLAMI 관련 정보

AWS Deep Learning AMIs 개발자 가이드 외부에서 DLAMI 관련 정보가 포함된 다른 리소스를 찾을 수 있습니다. 에서 다른 고객의 DLAMI에 대한 질문을 AWS re:Post확인하거나 직접 질문하세요. AWS Machine Learning 블로그 및 기타 AWS 블로그에서 DLAMI에 대한 공식 게시물을 읽습니다.

AWS re:Post

태그: AWS Deep Learning AMIs

AWS 블로그

- AWS Machine Learning 블로그 | 범주: AWS Deep Learning AMIs
- AWS Machine Learning 블로그 | Amazon EC2 C5 및 P3 인스턴스에서 최적화된 TensorFlow 1.6을 사용한 더 빠른 훈련
- AWS Machine Learning 블로그 | Machine Learning 실무자를 AWS Deep Learning AMIs 위한 새로운 기능
- AWS Partner Network (APN) 블로그 | 사용 가능한 새 교육 과정:의 Machine Learning 및 딥 러닝 소개 AWS
- AWS 뉴스 블로그 |를 통한 딥 러닝 여정 AWS

DLAMI의 사용 중단된 기능

다음 표에는 AWS Deep Learning AMIs (DLAMI)의 더 이상 사용되지 않는 기능, 더 이상 사용되지 않는 날짜, 더 이상 사용되지 않는 이유에 대한 세부 정보가 나와 있습니다.

Feature	날짜	세부 사항
Ubuntu 16.04	10/07/2021	Ubuntu Linux 16.04 LTS가 2021년 4월 20일자로 LTS 5년 기한이 만료됨에 따라 공급업체의 지원이 중단됩니다. 2021년 10월 기준 새 릴리스부터 딥 러닝 베이스 AMI(Ubuntu 16.04)에 대한 업데이트가 중단됩니다. 이전 릴리스는 계속 사용할 수 있습니다.
Amazon Linux	10/07/2021	Amazon Linux 는 2020년 12월부로 수명이 종료됩니다. 2021년 10월 기준 새 릴리스부터 Deep Learning AMI(Amazon Linux)에 대한 업데이트가 중단됩니다. Deep Learning AMI(Amazon Linux)의 이전 릴리스는 계속 사용할 수 있습니다.
Chainer	07/01/2020	Chainer는 2019년 12월부로 주요 릴리스의 종료를 공지했습니다. 따라서 2020년 7월부터 Chainer Conda 환경은 DLAMI에 더 이상 포함되어 있지 않습니다. 이러한 환경을 포함하는 DLAMI의 이전 릴리스는 계속 사용할 수 있습니다. 이러한 프레임워

Feature	날짜	세부 사항
		크에 대해 오픈 소스 커뮤니티에서 게시한 보안 수정 사항이 있는 경우에만 이러한 환경에 대한 업데이트를 제공합니다.
Python 3.6	06/15/2020	고객의 요청에 따라 새로운 TF/MX/PT 릴리스를 위해 Python 3.7로 이전하고 있습니다.
Python 2	01/01/2020	Python 오픈 소스 커뮤니티는 Python 2에 대한 지원을 공식적으로 종료했습니다. 또한 TensorFlow, PyTorch 및 MXNet 커뮤니티는 TensorFlow 1.15, TensorFlow 2.1, PyTorch 1.4 및 MXNet 1.6.0 릴리스가 Python 2를 지원하는 마지막 릴리스가 될 것이라고 공지했습니다.

DLAMI 관련 문서 기록

다음 표에는 최근 DLAMI 릴리스와 관련 AWS Deep Learning AMIs 개발자 가이드 변경 사항의 기록이 나와 있습니다.

최근 변경 사항

변경 사항	설명	날짜
TensorFlow Serving을 사용하여 MNIST 모델 훈련	Tensorflow 서빙을 사용하여 MNIST 모델을 훈련하는 예제입니다.	2025년 2월 14일
ARM64 DLAMI	는 AWS Deep Learning AMIs 이제 Arm64 프로세서 기반 GPUs에서 이미지를 지원합니다.	2021년 11월 29일
TensorFlow 2	Conda를 사용하는 Deep Learning AMI는 이제 CUDA 10을 사용하는 TensorFlow 2와 함께 제공됩니다.	2019년 12월 3일
AWS 추론	이제 딥 러닝 AMI가 AWS Inferentia 하드웨어와 AWS Neuron SDK를 지원합니다.	2019년 12월 3일
Nightly Build로 PyTorch 설치	PyTorch를 제거한 다음 Conda를 사용하는 DLAMI에 Nightly Build로 PyTorch를 설치하는 방법을 다루는 자습서가 추가되었습니다.	2018년 9월 25일
Conda 자습서	예제 MOTD가 업데이트되어 더욱 최신 릴리스를 반영했습니다.	2018년 7월 23일

이전 변경 사항

다음 표에는 2018년 7월 이전의 DLAMI 릴리스와 관련 변경 사항의 기록이 나와 있습니다.

변경 사항	설명	날짜
Horovod 포함 TensorFlow	TensorFlow 및 Horovod가 포함된 ImageNet 교육용 자습서가 추가되었습니다.	2018년 6월 6일
업그레이드 안내서	업그레이드 안내서가 추가되었습니다.	2018년 5월 15일
새로운 리전과 새로운 10분 자습서	새로운 리전인 미국 서부(캘리포니아 북부), 남아메리카, 캐나다(중부), EU(런던), EU(파리)가 추가되었습니다. 또한 10분 자습서의 첫 번째 릴리스 제목이 'Deep Learning AMI 시작하기'로 지정되었습니다.	2018년 4월 26일
Chainer 자습서	다중 GPU, 단일 GPU 및 CPU 모드의 Chainer 사용 자습서가 추가되었습니다. 여러 프레임워크에 대한 CUDA 통합이 CUDA 8에서 CUDA 9로 업그레이드되었습니다.	2018년 2월 28일
Linux AMIs v3.0과 함께 MXNet Model Server, TensorFlow Serving, TensorBoard 출시	MXNet Model Server v0.1.5, TensorFlow Serving v1.4.0, TensorBoard v0.4.0. AMI, 그리고 Conda 및 CUDA 개요에서 설명한 프레임워크 CUDA 기능을 이용하는 시각화 서비스 기능과 새로운 모델을 더하여 Conda AMI에 대한 자습서 추가. AMI와 프레임워크 CUDA 기능은 Conda 및 CUDA 개요	2018년 1월 25일

변경 사항	설명	날짜
	에 설명되어 있음. AMI 및 프레임워크 CUDA 기능은 Conda 및 CUDA 개요에 설명되어 있음. 최신 출시 정보를 https://aws.amazon.com/releasenotes/(으)로 이동	
Linux AMIs v2.0	NCCL 2.1로 Base, Source 및 Conda AMI 업데이트됨. MXNet 버전 1.0, PyTorch 0.3.0, 및 Keras 2.0.9.로 업데이트된 소스 및 Conda AMI.	2017년 12월 11일
2개의 Windows AMI 옵션 추가	Windows 2012 R2 및 2016 AMI 릴리스: AMI 선택 설명서에 추가 및 출시 정보에 추가.	2017년 11월 30일
최초 문서 릴리스	변경된 주제/단원의 링크를 포함하는 변경 사항에 대한 자세한 설명.	2017년 11월 15일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.