



개발자 안내서

AWS Deep Learning AMIs



AWS Deep Learning AMIs: 개발자 안내서

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

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

DLAMI란 무엇인가요?	1
이 가이드 정보	1
사전 조건	1
사용 사례 예제	1
특성	2
사전 설치된 프레임워크	2
GPU사전 설치된 소프트웨어	3
모델 제공 및 시각화	3
시작하기	4
시작하는 방법 DLAMI	4
DLAMI선택션	4
CUDA 설치 및 프레임워크 바인딩	5
Base	6
Conda	6
아키텍처	8
OS	8
인스턴스 선택	8
요금	10
리전 가용성	10
GPU	10
CPU	11
Inferentia	12
Trainium	13
런칭 a DLAMI	14
1단계: DLAMI 시작	14
DLAMIID 검색	15
아마존 EC2 콘솔에서 실행	16
2단계: DLAMI에 연결	17
3단계: 테스트 DLAMI	17
4단계: DLAMI 인스턴스 관리	18
정리	18
Jupyter 설정	18
Jupyter 보안	19
서버 시작	20

클라이언트 구성	20
Jupyter Notebook 서버에 로그인	22
a 사용 DLAMI	25
Conda DLAMI	25
Conda를 AMI 사용한 딥 러닝 소개	25
Your에 로그인하세요. DLAMI	26
TensorFlow 환경을 시작하십시오.	26
PyTorch Python 3 환경으로 전환하기	27
환경 제거	28
베이스 DLAMI	28
딥러닝 베이스 사용 AMI	28
CUDA버전 구성	28
Jupyter Notebook	29
설치된 자습서 탐색	30
Jupyter를 사용하여 환경 전환	30
자습서	31
프레임워크 활성화	31
Elastic Fabric Adapter	34
GPU모니터링 및 최적화	48
AWS Inferentia	58
ARM64 DLAMI	80
Inference	83
모델 제공	83
DLAMI 업그레이드	89
DLAMI 업그레이드	89
소프트웨어 업데이트	90
출시 알림	90
보안	92
데이터 보호	92
ID 및 액세스 관리	93
자격 증명을 통한 인증	94
정책을 사용하여 액세스 관리	97
IAM아마존과 함께 EMR	99
로깅 및 모니터링	99
사용량 추적	99
규정 준수 검증	100

복원력	100
인프라 보안	101
프레임워크 지원 정책	102
DLAMI프레임워크 지원 FAQs	102
어떤 프레임워크 버전에 보안 패치가 적용되나요?	103
이미지는 어떤 역할을 하나요? AWS 새 프레임워크 버전이 출시되면 게시되나요?	103
새로 추가되는 이미지/ SageMaker AWS 특징?	103
지원되는 프레임워크 포에서 현재 버전을 어떻게 확인하나요?	103
지원되는 프레임워크 포에 없는 버전을 실행 중인 경우에는 어떻게 해야하나요?	103
의 이전 버전을 DLAMIs 지원하나요 TensorFlow?	103
지원되는 프레임워크 버전에 대해 최신 패치가 적용된 이미지를 찾으려면 어떻게 해야 하나 요?	104
새 이미지는 얼마나 자주 릴리스되나요?	104
워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요?	104
패치가 적용되거나 업데이트된 새 프레임워크 버전을 사용할 수 있게 되면 어떻게 해야 하나 요?	104
프레임워크 버전을 변경하지 않고도 종속성이 업데이트되나요?	104
내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?	105
더 이상 활발하게 유지 관리되지 않는 프레임워크 버전의 이미지도 패치가 적용되나요?	106
이전 프레임워크 버전을 사용하고 싶습니다.	106
프레임워크 및 해당 버전의 지원 변경 사항을 계속 up-to-date 확인하려면 어떻게 해야 합니 까?	106
Anaconda 리포지토리를 사용하려면 상용 라이선스가 필요한가요?	107
중요 변경사항	108
DLAMINVIDIA드라이버 변경 FAQs	108
무엇이 바뀌었나요?	108
이 변경이 필요한 이유는 무엇인가요?	109
이번 변경은 어떤 영향을 미쳤습니까? DLAMIs	109
이것이 여러분에게 어떤 의미가 있나요?	110
최신 버전에서 기능 손실이 발생합니까? DLAMIs	110
이 변경 사항이 Deep Learning Containers에 영향을 미쳤습니까?	110
관련 정보	111
릴리스 정보	112
베이스 DLAMIs	112
단일 프레임워크 DLAMIs	113
멀티 프레임워크 DLAMIs	114

더 이상 사용되지 않는 기능	115
문서 기록	117
.....	CXX

무엇입니까 AWS Deep Learning AMIs?

AWS Deep Learning AMIs (DLAMI) 는 클라우드에서 딥 러닝에 사용할 수 있는 사용자 지정된 기계 이미지를 제공합니다. 대부분의 지역에서 사용할 수 있습니다. DLAMIs AWS 리전 소규모 CPU 전용 인스턴스부터 고성능 최신 멀티 GPU 인스턴스에 이르기까지 다양한 Amazon Elastic Compute Cloud (AmazonEC2) 인스턴스 유형에 사용할 수 있습니다. 여기에는 가장 널리 사용되는 딥 러닝 [NVIDIADNN프레임워크의 최신 NVIDIA CUDA 릴리스와 cu가](#) 사전 구성되어 DLAMIs 제공됩니다.

이 가이드 정보

에 있는 내용은 시작 및 사용에 도움이 될 수 있습니다. DLAMIs 이 가이드에서는 트레이닝과 추론 모두에 대한 몇 가지 일반적인 딥 러닝 사용 사례를 다룹니다. 또한 목적에 AMI 맞는 것을 선택하는 방법과 선호할 수 있는 인스턴스의 종류도 다룹니다.

또한 DLAMIs 여기에는 지원되는 프레임워크에서 제공하는 여러 자습서가 포함되어 있습니다. 이 가이드에서는 각 프레임워크를 활성화하는 방법과 시작하는 데 적합한 자습서를 찾는 방법을 보여 줍니다. 또한 분산 교육, 디버깅, 사용에 대한 자습서도 포함되어 있습니다. AWS 추론 및 AWS 트레이니움 및 기타 주요 개념. 브라우저에서 자습서를 실행하도록 Jupyter 노트북 서버를 설정하는 방법에 대한 지침은 을 참조하십시오. [Jupyter Notebook 서버 설정](#)

사전 조건

를 성공적으로 실행하려면 명령줄 도구와 기본 Python에 익숙해지는 것이 좋습니다. DLAMIs

DLAMI 사용 사례 예시

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

딥러닝에 대한 DLAMI 학습은 머신 러닝과 딥 러닝 프레임워크를 학습하거나 교육하기 위한 훌륭한 선택입니다. 이를 통해 DLAMIs 각 프레임워크의 설치 문제를 해결하고 동일한 컴퓨터에서 실행되도록 하는 번거로움을 없앨 수 있습니다. Jupyter 노트북이 DLAMIs 포함되어 있어 기계 학습과 딥 러닝을 처음 접하는 사용자에게 프레임워크가 제공하는 자습서를 쉽게 실행할 수 있습니다.

앱 개발 — 딥 러닝을 사용하여 앱이 AI의 최신 기술을 활용하도록 만드는 데 관심이 있는 앱 개발자라면 이 제품이 완벽한 DLAMI 테스트베드입니다. 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 러닝으로 데이터를 처리하는 데 관심이 있다면 많은 프레임워크가 R 및 Spark를 지원한다는 것을 알게 될 것입니다. 개인화 및 예측 시스템에 대한 확장형 데이터 처리 시스템 빌드에 이르는 단순 회귀 작업을 수행하는 방법에 관한 자습서를 찾을 수 있습니다.

연구 — 새로운 프레임워크를 시험해 보거나, 새 모델을 테스트하거나, 새 모델을 학습시키려는 연구자라면, DLAMI AWS 확장 기능을 사용하면 여러 교육 노드를 설치하고 관리하는 번거로운 작업을 줄일 수 있습니다.

Note

첫 번째 선택은 인스턴스 유형을 더 많은 인스턴스 GPUs (최대 8개)가 포함된 더 큰 인스턴스로 업그레이드하는 것이지만, 인스턴스 클러스터를 생성하여 수평적으로 확장할 수도 있습니다. DLAMI 클러스터 빌드에 대한 자세한 내용은 [에 대한 관련 정보 DLAMI](#) 섹션을 참조하세요.

DLAMI의 기능

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

사전 설치된 프레임워크


운영 체제 (OS) 및 소프트웨어 DLAMI 버전과 관련된 다른 변형에는 현재 두 가지 주요 버전이 있습니다.

- [AMIConda를 사용한 딥 러닝](#)— conda 패키지와 별도의 Python 환경을 사용하여 프레임워크를 별도로 설치합니다.
- [Deep Learning Base AMI](#)— 프레임워크가 설치되지 않고 다른 종속성만 설치되어 [NVIDIA CUDA](#) 있습니다.

Deep Learning AMI with Conda는 conda 환경을 사용하여 각 프레임워크를 분리하므로 원하는 대로 프레임워크를 전환할 수 있으며 종속성 충돌을 걱정할 필요가 없습니다. Conda를 AMI 사용한 딥 러닝은 다음 프레임워크를 지원합니다.

- PyTorch

- TensorFlow 2.

 Note

DLAMI아파치, MXNet Microsoft Cognitive Toolkit (CNTK), Caffe, Caffe, Caffe2, Theano, Chainer, Keras와 같은 딥 러닝 프레임워크는 더 이상 지원하지 않습니다.

GPU사전 설치된 소프트웨어

CPU인스턴스만 사용하는 경우에도 계속 사용할 수 [NVIDIACUDA](#)있습니다 [NVIDIA. DLAMIs DNN](#) 설치된 소프트웨어는 인스턴스 유형에 관계없이 동일합니다. GPU특정 도구는 하나 GPU 이상의 인스턴스에서만 작동한다는 점에 유의하세요. 인스턴스 유형에 대한 자세한 내용은 [을 참조하십시오DLAMI 인스턴스 유형 선택](#).

에 대한 자세한 내용은 CUDA [을 참조하십시오CUDA 설치 및 프레임워크 바인딩](#).

모델 제공 및 시각화

AMIConda를 사용한 딥 러닝에는 모델 시각화뿐만 아니라 모델 TensorBoard 시각화용 TensorFlow 모델 서버가 사전 설치되어 있습니다. 자세한 내용은 [TensorFlow 서빙](#) 단원을 참조하십시오.

시작하기

시작하는 방법 DLAMI

이 가이드에는 적합한 인스턴스 선택, 사용 사례 및 예산에 맞는 인스턴스 유형 선택, 관심 있을 [에 대한 관련 정보 DLAMI](#) 만한 사용자 지정 설정 설명 등에 대한 팁이 포함되어 있습니다. DLAMI

처음 사용하는 경우 AWS Amazon을 사용하는 EC2 경우 부터 시작하십시오 [AMIConda를 사용한 딥 러닝](#). Amazon EC2 및 기타 제품에 대해 잘 알고 있다면 AWS AmazonEMR, Amazon 또는 Amazon EFS S3와 같은 서비스에서 분산 교육 또는 추론이 필요한 프로젝트에 이러한 서비스를 통합하는 데 관심이 있다면 사용 [에 대한 관련 정보 DLAMI](#) 사례에 맞는 지 확인해 보십시오.

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

다음 단계

[선택: DLAMI](#)

선택: DLAMI

다양한 DLAMI 옵션을 제공합니다. 사용 DLAMI 사례에 맞는 올바른 이미지를 선택할 수 있도록 개발된 하드웨어 유형 또는 기능별로 이미지를 그룹화합니다. 최상위 그룹 분류는 다음과 같습니다.

- DLAMI 유형: 기본 CUDA 프레임워크와 단일 프레임워크 대 다중 프레임워크 (Conda) 비교 DLAMI
- 컴퓨팅 아키텍처: x86 기반 대 ARM64 기반 기반 [AWS 그래비톤](#)
- 프로세서 유형: 추론 대 [GPU 트레이닝 CPU 대 인퍼런스](#)
- SDK [CUDA](#): vs. [AWS 뉴런](#)
- OS: Amazon Linux vs. Ubuntu

이 설명서의 나머지 섹션에서 유용한 세부 정보를 확인하세요.

주제

- [CUDA 설치 및 프레임워크 바인딩](#)
- [Deep Learning Base AMI](#)

- [AMIConda를 사용한 딥 러닝](#)
- [DLAMI 아키텍처 옵션](#)
- [DLAMI 운영 체제 옵션](#)

다음

[AMIConda를 사용한 딥 러닝](#)

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 지원하는 최신 버전을 사용하여 CUDA를 사용하는 모든 프레임워크에 대한 공식 pip 바이너리가 있습니다. 최신 버전을 사용하여 딥 러닝 환경을 사용자 지정하려는 경우 Deep Learning Base AMI를 사용하세요.

자세한 내용은 [안정적 후보와 릴리스 후보 비교](#)의 지침을 참조하십시오.

CUDA를 사용하는 DLAMI 선택

[Deep Learning Base AMI](#)에는 사용 가능한 모든 CUDA 버전 시리즈가 있습니다.

사용 가능한 모든 CUDA 버전 [AMIConda를 사용한 딥 러닝](#) 시리즈가 있습니다.

Note

MXNet, CNTK, Caffe, Caffe2, Theano, Chainer 또는 Keras Conda 환경은 더 이상 포함되지 않습니다. AWS Deep Learning AMIs

특정 프레임워크 버전 번호는 [DLAMIs 릴리스 정보](#) 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 [딥러닝 베이스 사용 AMI](#) 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 주제

- [Deep Learning Base AMI 사용](#)

AMIConda를 사용한 딥 러닝

DLAMIConda는 conda 가상 환경을 사용하며 다중 프레임워크 또는 단일 프레임워크로 제공됩니다. DLAMIs 이러한 환경을 다른 프레임워크 설치를 별도로 유지하고 프레임워크간 전환을 간소화하도록 구성됩니다. 제공되는 모든 프레임워크를 배우고 실험해 보는 데 아주 좋습니다. DLAMI 대부분의 사용자는 Conda를 AMI 사용한 새로운 딥 러닝이 자신에게 딱 맞다고 생각합니다.

프레임워크의 최신 버전으로 자주 업데이트되며 최신 GPU 드라이버와 소프트웨어가 포함되어 있습니다. 일반적으로 다음과 같이 불립니다. AWS 대부분의 AMIs 문서에는 딥러닝이 있습니다. 이들은 우분투 20.04, 아마존 리눅스 2 운영 체제를 DLAMIs 지원합니다. 운영 체제 지원은 업스트림 OS의 지원에 따라 달라집니다.

안정적 후보와 릴리스 후보 비교

Conda는 각 프레임워크의 최신 공식 릴리스의 최적화된 바이너리를 AMIs 사용합니다. 릴리스 후보 기능과 시험 사용 기능은 발표 예정이 없습니다. 최적화는 C5 및 C4 인스턴스 유형에 대한 학습 및 추론 속도를 높이는 Intel과 같은 가속 기술에 대한 프레임워크의 MKL DNN 지원에 따라 달라집니다. CPU 또한 바이너리는 AVX-2AVX, .1, .2를 포함하나 이에 국한되지 않는 고급 인텔 명령어 세트를 지원하도록 컴파일됩니다. SSE4 SSE4 이를 통해 인텔 CPU 아키텍처에서 벡터 및 부동 소수점 연산이 가속화됩니다. 또한, GPU 예를 들어, 유형의 경우 CUDA 와 DNN cu는 최신 공식 릴리스가 지원하는 버전으로 업데이트됩니다.

Conda를 AMI 사용한 딥 러닝은 프레임워크가 처음 활성화될 때 Amazon EC2 인스턴스에 가장 최적화된 버전의 프레임워크를 자동으로 설치합니다. 자세한 정보는 [Conda를 통한 딥 러닝 사용 AMI](#) 섹션을 참조하세요.

사용자 지정 옵션이나 최적화된 빌드 옵션으로 소스에서 직접 설치하려는 경우에는 [Deep Learning Base AMI](#)가 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 발표했습니다. Python 2 Conda 환경을 포함하는 이전 릴리스 DLAMI (v26, v25 등)는 계속 사용할 수 있습니다. 그러나 이전에 게시된 DLAMI 버전의 Python 2 Conda 환경 업데이트는 해당 버전에 대한 오픈 소스 커뮤니티에서 게시한 보안 수정이 있는 경우에만 제공합니다. DLAMI TensorFlow 및 PyTorch 프레임워크의 최신 버전이 포함된 릴리스에는 Python 2 Conda 환경이 포함되어 있지 않습니다.

CUDASupport

구체적인 CUDA 버전 번호는 [GPUDLAMI 릴리스 노트에서](#) 확인할 수 있습니다.

다음

[DLAMI 아키텍처 옵션](#)

관련 항목

- AMIConda와 함께 딥러닝을 사용하는 방법에 대한 [Conda를 통한 딥 러닝 사용 AMI](#) 자습서는 자습서를 참조하십시오.

DLAMI 아키텍처 옵션

AWS Deep Learning AMIs는 [x86 기반 또는 ARM64 기반 그라비톤2 아키텍처와 함께 제공됩니다.](#)[AWS](#)

ARM64 GPU DLAMI를 시작하는 방법에 대한 자세한 내용은 [을 참조하십시오.](#) [더 ARM64 DLAMI](#) 사용 가능한 인스턴스 유형에 대한 자세한 내용은 [DLAMI 인스턴스 유형 선택](#)을 참조하세요.

다음

[DLAMI 운영 체제 옵션](#)

DLAMI 운영 체제 옵션

DLAMI는 다음 운영 체제에서 제공됩니다.

- Amazon Linux 2
- 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 중단](#)을 참조하세요.

DLAMI를 선택하기 전에 필요한 인스턴스 유형을 평가하고 AWS 리전을 지정하세요.

다음

[DLAMI 인스턴스 유형 선택](#)

DLAMI 인스턴스 유형 선택

보다 일반적으로, 인스턴스 유형을 선택할 때는 다음 사항을 고려하십시오DLAMI.

- 딥 러닝을 처음 사용하는 경우 단일 인스턴스가 요구 사항에 적합할 GPU 수 있습니다.
- 예산이 한정되어 있다면 CPU 인스턴스만 사용할 수 있습니다.
- 딥 러닝 모델 추론을 위해 고성능 및 비용 효율성을 최적화하려는 경우 다음과 같은 인스턴스를 사용할 수 있습니다. AWS 인퍼런시아 칩.

- ARM64 기반 CPU 아키텍처의 고성능 GPU 인스턴스를 찾고 있다면 G5G 인스턴스 유형을 사용할 수 있습니다.
- 추론 및 예측을 위해 사전 학습된 모델을 실행하려는 경우 [Amazon Elastic Inference](#)를 [Amazon](#) 인스턴스에 연결할 수 있습니다. EC2 Amazon Elastic Inference를 사용하면 a 미만의 비용으로 액셀러레이터에 액세스할 수 있습니다. GPU
- 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. 모델을 클러스터에 배포할 수도 있습니다. GPUs 배치 크기를 줄이는 경우 적은 메모리를 가진 인스턴스를 사용하는 것이 더 나은 방법일 수 있습니다. 이는 정확도 및 훈련 속도에 영향을 줄 수 있습니다.
- 대규모 노드 간 통신이 필요한 NVIDIA 집단 통신 라이브러리 (NCCL) 를 사용하여 기계 학습 애플리케이션을 실행하려면 [Elastic Fabric Adapter \(EFA\)](#) 를 사용하는 것이 좋습니다.

인스턴스에 대한 자세한 내용은 인스턴스 참조하십시오.

다음 주제에서는 인스턴스 유형 고려 사항에 대한 정보를 제공합니다.

Important

AMIs 디플러닝에는 NVIDIA Corporation에서 개발, 소유 또는 제공한 드라이버, 소프트웨어 또는 툴킷이 포함됩니다. 귀하는 NVIDIA 하드웨어가 포함된 Amazon EC2 인스턴스에서만 이러한 NVIDIA 드라이버, 소프트웨어 또는 툴킷을 사용하는 데 동의합니다.

주제

- [DLAMI 요금](#)
- [DLAMI 리전 가용성](#)
- [권장 GPU 인스턴스](#)
- [권장 CPU 인스턴스](#)
- [권장 Inferentia 인스턴스](#)
- [권장 Trainium 인스턴스](#)

DLAMI 요금

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

일부 Amazon EC2 인스턴스 유형에는 무료라는 라벨이 붙어 있습니다. 이러한 무료 인스턴스 중 DLAMI 하나에서 를 실행할 수 있습니다. 즉, 해당 DLAMI 인스턴스의 용량만 사용할 경우 를 완전히 무료로 사용할 수 있습니다. 더 많은 CPU 코어, 더 많은 디스크 공간, 더 많은 또는 하나 이상의 더 RAM 강력한 인스턴스가 필요한 경우에는 프리 티어 인스턴스 클래스에 속하지 않는 인스턴스가 필요합니다. GPUs

인스턴스 선택 및 요금에 대한 자세한 내용은 [Amazon EC2 요금을](#) 참조하십시오.

DLAMI 리전 가용성

각 리전은 다양한 범위의 인스턴스를 지원하며 인스턴스 유형별 비용은 리전마다 달라질 수 있습니다. DLAMIs 모든 지역에서 사용할 수 있는 것은 아니지만 원하는 지역으로 DLAMIs 복사할 수 있습니다. 자세한 내용은 [복사를 AMI](#) 참조하십시오. 리전 선택 목록을 참고하고 사용자와 고객에게 가까운 리전을 선택하세요. 둘 DLAMI 이상을 사용하고 클러스터를 생성할 가능성이 있는 경우 클러스터의 모든 노드에 동일한 지역을 사용해야 합니다.

지역에 대한 자세한 내용은 지역 참조하십시오.

다음

[권장 GPU 인스턴스](#)

권장 GPU 인스턴스

대부분의 딥 러닝 용도에는 GPU 인스턴스를 사용하는 것이 좋습니다. 새 모델을 학습시키는 것이 GPU 인스턴스보다 인스턴스에서 더 빠릅니다. CPU 다중 GPU 인스턴스가 있거나 를 사용하여 여러 인스턴스에 분산 학습을 사용하는 경우 하위 선형적으로 확장할 수 있습니다. GPUs

다음 인스턴스 유형은 를 지원합니다. DLAMI GPU인스턴스 유형 옵션 및 사용에 대한 자세한 내용은 참조하고 액셀러레이티드 컴퓨팅을 선택하십시오.

Note

모델의 크기는 인스턴스를 선택하는 요소로 사용됩니다. 모델의 인스턴스 가용 용량을 초과할 경우 애플리케이션에 충분한 메모리가 있는 다른 인스턴스 유형을 선택하십시오. RAM

- [아마존 EC2 P5e 인스턴스](#)는 최대 8개의 NVIDIA 테슬라 H200을 지원합니다. GPUs
- [아마존 EC2 P5 인스턴스](#)는 최대 8개의 NVIDIA 테슬라 GPU H100을 지원합니다.
- [아마존 EC2 P4 인스턴스](#)는 최대 8개의 NVIDIA 테슬라 GPU A100을 지원합니다.
- [아마존 EC2 P3 인스턴스](#)는 최대 8개의 NVIDIA 테슬라 GPU V100을 지원합니다.
- [Amazon EC2 G3 인스턴스](#)는 최대 4개의 NVIDIA 테슬라 GPU M60을 지원합니다.
- [Amazon EC2 G4 인스턴스](#)는 NVIDIA GPU T4를 최대 4개까지 보유합니다.
- [Amazon EC2 G5 인스턴스](#)는 최대 8개의 NVIDIA GPU A10G를 지원합니다.
- [Amazon EC2 G6 인스턴스](#)는 최대 8개의 NVIDIA GPU L4를 지원합니다.
- [Amazon EC2 G6e 인스턴스](#)에는 최대 8개의 NVIDIA L40S 텐서 코어가 있습니다. GPUs
- [Amazon EC2 G5G 인스턴스](#)에는 ARM64 기반이 있습니다. [AWS](#) 그라비톤2 프로세서.

DLAMI 인스턴스는 프로세스를 모니터링하고 최적화하는 도구를 제공합니다. GPU 프로세스 모니터링에 대한 자세한 내용은 [GPU 모니터링 및 최적화](#)를 참조하십시오.

G5g 인스턴스 사용에 대한 자세한 내용은 [더 ARM64 DLAMI](#) 자습서를 참조하세요.

다음

[권장 CPU 인스턴스](#)

권장 CPU 인스턴스

예산이 한정되어 있든, 딥 러닝에 대해 배우고 있든, 예측 서비스를 실행하려는 경우든, CPU 카테고리에는 저렴한 옵션이 많이 있습니다. 일부 프레임워크는 C5 (일부 지역에서만 사용 가능) CPU 인스턴스 유형에 대한 학습 및 추론 속도를 높이는 Intel의 MKL DNN 이점을 활용합니다. 인스턴스 유형에 대한 자세한 내용은 CPU 인스턴스 유형, 참조하고 컴퓨팅 최적화를 선택하십시오.

Note

모델의 크기는 인스턴스를 선택하는 요소로 사용됩니다. 모델의 인스턴스 가용 용량을 초과할 경우 애플리케이션에 충분한 메모리가 있는 다른 인스턴스 유형을 선택하십시오. RAM

- [Amazon EC2 C5 인스턴스](#)에는 최대 72개의 vCPUs 인텔이 있습니다. C5 인스턴스는 과학적 모델링, 배치 처리, 분산 분석, 고성능 컴퓨팅 (HPC), 기계 및 딥 러닝 추론에 탁월합니다.

다음

[권장 Inferentia 인스턴스](#)

권장 Inferentia 인스턴스

AWS Inferentia 인스턴스는 딥 러닝 모델 추론 워크로드에 고성능 및 비용 효율성을 제공하도록 설계되었습니다. 특히 Inf2 인스턴스 유형에서는 다음을 사용합니다. AWS 인퍼런시아 칩과 [AWS Neuron](#)은 [SDK](#) 및 와 같은 인기 있는 기계 학습 프레임워크와 통합됩니다. TensorFlow PyTorch

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 크기는 인스턴스를 선택하는 요소로 사용됩니다. 모델에서 사용 가능한 RAM 인스턴스를 초과하는 경우 애플리케이션에 충분한 메모리가 있는 다른 인스턴스 유형을 선택하십시오.

- [아마존 EC2 Inf2 인스턴스](#)는 최대 16개까지 가능 AWS 인퍼런시아 칩과 100Gbps의 네트워킹 처리량.

시작하는 방법에 대한 자세한 내용은 AWS 추론은 DLAMIs 을 참조하십시오. [The AWS 인퍼런시아 칩 너비 DLAMI](#)

다음

[권장 Trainium 인스턴스](#)

권장 Trainium 인스턴스

AWS Trainium 인스턴스는 딥 러닝 모델 추론 워크로드에 고성능 및 비용 효율성을 제공하도록 설계되었습니다. 특히 Trn1 인스턴스 유형에서는 다음을 사용합니다. AWS 트레이니엄 칩과 [AWS Neuron은 SDK](#) 및 와 같은 인기 있는 기계 학습 프레임워크와 통합됩니다. TensorFlow PyTorch

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 크기는 인스턴스를 선택하는 요소로 사용됩니다. 모델에서 사용 가능한 RAM 인스턴스를 초과하는 경우 애플리케이션에 충분한 메모리가 있는 다른 인스턴스 유형을 선택하십시오.

- [Amazon EC2 Trn1 인스턴스](#)는 최대 16개까지 AWS 트레이니엄 칩과 100Gbps의 네트워킹 처리량.

DLAMI 시작 및 구성

여기에 AMI 계시다면 어떤 제품을 출시할지 이미 잘 알고 계실 것입니다. 그렇지 않은 경우 에서 DLAMI a와 관련 하드웨어, 프레임워크 및 ID 검색을 찾아보십시오. [DLAMIs 릴리스 정보](#)

선택할 인스턴스 유형과 리전을 알고 있어야 합니다. 그렇지 않은 경우 [DLAMI 인스턴스 유형 선택](#)을 검색하세요.

Note

예시에서는 p3.16xlarge를 기본 인스턴스 유형으로 사용합니다. 원하는 인스턴스 유형으로 대체할 수 있습니다.

Important

Elastic Inference를 사용하려는 경우, Elastic [Inference를 실행하기 전에 먼저 Elastic Inference](#) 설정을 완료해야 합니다. DLAMI

주제

- [1단계: DLAMI 시작](#)
- [2단계: DLAMI에 연결](#)
- [3단계: 테스트 DLAMI](#)
- [4단계: DLAMI 인스턴스 관리](#)
- [정리](#)
- [Jupyter Notebook 서버 설정](#)

1단계: DLAMI 시작

Note

이 안내를 위해 딥러닝 AMI (Ubuntu 18.04) 과 관련된 참고 자료를 만들 수 있을 것입니다. 다른 DLAMI 것을 선택하더라도 이 가이드를 따를 수 있을 것입니다.

1. [본인의 ID를 찾아보세요. DLAMI](#)
2. [다음에서 Amazon EC2 인스턴스를 시작합니다. DLAMI](#)

Amazon EC2 콘솔을 사용하게 됩니다. [아마존 EC2 콘솔에서 실행](#)에 안내된 자세한 지침을 따르세요.

Tip

CLI 옵션: 를 DLAMI 사용하여 스펀업하기로 선택한 경우 AWS CLIAMI의 ID, 지역 및 인스턴스 유형, 보안 토큰 정보가 필요합니다. AMI 및 인스턴스가 IDs 준비되어 있는지 확인하세요. 아직 설정하지 않은 경우 AWS CLI 아직 [설치 안내서를 사용하여 먼저 설정하세요. AWS 명령줄 인터페이스.](#)

3. 이러한 옵션 중 하나를 통해 단계를 완료한 이후 인스턴스가 준비될 때까지 기다리세요. 이는 보통 몇 분 정도 소요됩니다. [EC2 콘솔에서](#) 인스턴스의 상태를 확인할 수 있습니다.

DLAMI ID 검색

각각은 고유한 식별자 (ID) 를 AMI 가지고 있습니다. 다음을 사용하여 원하는 ID에 DLAMI 대해 쿼리할 수 있습니다. AWS 명령줄 인터페이스 (AWS CLI). 아직 가지고 있지 않은 경우 AWS CLI 설치하려면 [시작하기 를 참조하십시오. AWS CLI.](#)

Note

추가 소프트웨어 구성 (드라이버, python 버전, Amazon EBS 유형) 은 의 DLAMI 릴리스 노트를 참조하십시오. [DLAMIs 릴리스 정보](#)

1. 다음 사항을 확인하십시오. AWS 자격 증명이 구성되어 있습니다.

```
aws configure
```

2. 다음 명령을 사용하여 사용자 ID를 DLAMI 검색하거나 에서 제공된 쿼리를 찾을 수 있습니다. AWS Deep Learning AMIs 릴리스 노트.

```
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, &CreationDate))[:1].ImageId' --output text
```

Note

지정된 프레임워크의 릴리스 버전을 지정하거나 버전 번호를 물음표로 대체하여 최신 릴리스를 가져올 수 있습니다.

3. 다음과 같이 출력됩니다

```
ami-09ee1a996ac214ce7
```

이 DLAMI ID를 복사한 다음 버튼을 q 눌러 프롬프트를 종료합니다.

다음 단계

[아마존 EC2 콘솔에서 실행](#)

아마존 EC2 콘솔에서 실행

Note

Elastic Fabric Adapter (EFA) 로 인스턴스를 시작하려면 [다음 단계를 참조하십시오.](#)

1. [EC2콘솔](#)을 엽니다.
2. 최상위 탐색의 현재 리전을 기록합니다. 원하는 것이 아닌 경우 AWS 리전계속하기 전에 이 옵션을 변경하세요. 자세한 내용은 참조하십시오.
3. 인스턴스 시작을 선택합니다.
4. 인스턴스 이름을 입력하고 DLAMI 적합한 이름을 선택합니다.
 - a. DLAMIMy에서 기존 항목을 AMIs찾거나 Quick Start를 선택합니다.
 - b. DLAMIID로 검색하세요. 옵션을 찾은 후 선택합니다.
5. 인스턴스 유형을 선택합니다. 권장 인스턴스 패밀리는 DLAMI 다음에서 찾을 수 있습니다. AWS Deep Learning AMIs 릴리스 노트. DLAMI인스턴스 유형에 대한 일반적인 권장 사항은 [인스턴스 선택](#)을 참조하십시오.

Note

[Elastic Inference](#)(EI)를 사용하려면 인스턴스 세부 정보 구성을 클릭하고 Amazon EI 가속기 추가를 선택한 후 Amazon EI 가속기의 크기를 선택합니다.

6. 인스턴스 시작을 선택합니다.

Tip

를 사용하여 딥러닝을 [시작하기를 확인하세요. AWS 스크린샷을 이용한 단계별 설명을 AMI 위한 딥러닝](#)

다음 단계

[2단계: DLAMI에 연결](#)

2단계: DLAMI에 연결

클라이언트 (윈도우, macOS 또는 리눅스) 에서 실행한 파일에 연결합니다. DLAMI 자세한 내용은 Amazon EC2 사용 설명서의 [Linux 인스턴스에 연결](#)을 참조하십시오.

로그인한 후 Jupyter 설정을 수행하려면 SSH 로그인 명령어 사본을 준비해 두십시오. 변형을 사용하여 Jupyter 웹페이지에 연결합니다.

다음 단계

[3단계: 테스트 DLAMI](#)

3단계: 테스트 DLAMI

DLAMI버전에 따라 다양한 테스트 옵션이 있습니다.

- [AMIConda를 사용한 딥러닝](#) - [Conda를 통한 딥러닝 사용 AMI](#)로 이동합니다.
- [Deep Learning Base AMI](#) - 원하는 프레임워크의 설치 설명서를 참조하세요.

또한 Jupyter Notebook을 생성하고, 자습서를 시험해 보고, Python에서 코딩을 시작할 수 있습니다. 자세한 내용은 [Jupyter Notebook 서버 설정](#) 단원을 참조하십시오.

4단계: DLAMI 인스턴스 관리

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

Amazon Linux 또는 Ubuntu를 사용하는 경우 로그인하면 업데이트가 있는지 여부와 업데이트 지침을 참조하십시오. DLAMI Amazon Linux 유지 관리에 대한 자세한 내용은 [인스턴스 소프트웨어 업데이트](#) 섹션을 참조하세요. Ubuntu 인스턴스는 공식 [Ubuntu 설명서](#)를 참조하십시오.

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

Important

Meltdown 및 Spectre 취약성과 이러한 취약성을 해결하기 위해 운영 체제를 패치하는 방법에 대한 자세한 내용은 보안 게시판을 참조하십시오. [AWS-2018-013](#).

정리

더 이상 필요하지 않은 DLAMI 경우 계속 요금이 청구되지 않도록 중지하거나 종료할 수 있습니다. 인스턴스 중단은 상태가 계속 유지되어 나중에 다시 시작할 수 있습니다. 구성, 파일 및 기타 비휘발성 정보가 Amazon S3의 볼륨에 저장됩니다. 인스턴스가 중지된 동안 볼륨 유지를 위한 소규모의 S3 비용이 부과되지만 중단된 상태에 있는 동안 컴퓨팅 리소스에 대한 비용은 부과되지 않습니다. 인스턴스를 다시 시작할 때 볼륨이 탑재되고 데이터가 볼륨에 위치합니다. 인스턴스를 종료한 경우 이는 사라져 다시 시작할 수 없습니다. 데이터는 여전히 S3에 상주하지만, 추가 요금을 방지하기 위해 볼륨 또한 삭제해야 합니다. 자세한 지침은 Amazon 사용 EC2 설명서의 [인스턴스 종료를](#) 참조하십시오.

Jupyter Notebook 서버 설정

Jupyter Notebook 서버를 사용하면 DLAMI 인스턴스에서 Jupyter Notebook을 만들고 실행할 수 있습니다. Jupyter Notebook을 사용하면 AWS 인프라를 사용하고 DLAMI에 내장된 패키지에 액세스하면서 학습 및 추론을 위한 기계 학습(ML) 실험을 수행할 수 있습니다. Jupyter Notebook에 대한 자세한 내용은 [Jupyter Notebook 문서](#)를 참조하세요.

Jupyter Notebook 설정 요건:

- Amazon EC2 DLAMI 인스턴스에서 Jupyter Notebook 서버를 구성합니다.

- Jupyter Notebook 서버에 연결할 수 있도록 클라이언트를 구성합니다. Windows, macOS 및 Linux 클라이언트에 대한 구성 지침이 제공됩니다.
- Jupyter Notebook 서버에 로그인하여 설정을 테스트합니다.

다음 섹션을 참조하여 Jupyter 설정 단계를 완료하세요. Jupyter Notebook 서버를 설정한 후에는 DLAMI와 함께 제공되는 예제 노트북 실행에 대한 자세한 내용을 [Jupyter Notebook 자습서 실행](#)에서 확인하세요.

주제

- [Jupyter 서버 보안](#)
- [Jupyter Notebook 서버 시작](#)
- [Jupyter 서버에 연결하도록 클라이언트 구성](#)
- [Jupyter Notebook 서버에 로그인하여 설정 테스트](#)

Jupyter 서버 보안

여기에서는 SSL과 사용자 지정 암호로 Jupyter를 설정합니다.

Amazon EC2 인스턴스에 연결하고 다음 단계를 완료하세요.

Jupyter 서버 구성

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

타사 서명된, 브라우저에서 보안 경고를 제공하지 않는 일반 SSL 인증서를 만들기를 원할 수 있습니다. 이 과정은 훨씬 더 복잡합니다. 자세한 정보는 [Jupyter 문서](#)를 참조하십시오.

다음 단계

[Jupyter Notebook 서버 시작](#)

Jupyter Notebook 서버 시작

이제 인스턴스에 로그인하고 이전 단계에서 작성한 SSL 인증서를 사용하는 다음 명령을 실행하여 Jupyter 서버를 시작할 수 있습니다.

```
$ 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 localhost URL을 통해 서버에 액세스할 수 있다는 콜아웃을 무시하십시오. 터널을 만들기 전에는 작동하지 않기 때문입니다.

Note

Jupyter 웹 인터페이스를 사용하여 프레임워크를 전환할 때 Jupyter가 환경 변경을 처리합니다. 이에 대한 자세한 내용은 [Jupyter를 사용하여 환경 전환](#)에서 확인하실 수 있습니다.

다음 단계

[Jupyter 서버에 연결하도록 클라이언트 구성](#)

Jupyter 서버에 연결하도록 클라이언트 구성

Jupyter Notebook 서버에 연결하도록 클라이언트를 구성한 후 작업 영역의 서버에서 노트북을 생성 및 액세스하고, 서버에서 딥 러닝 코드를 실행할 수 있습니다.

구성 정보에 대해서는 다음 링크 중 하나를 선택하십시오.

주제

- [Windows 클라이언트 구성](#)
- [Linux 또는 macOS 클라이언트 구성](#)

Windows 클라이언트 구성

Prepare

SSH 터널 설정에 필요한 다음 정보가 있어야 합니다.

- Amazon EC2 인스턴스의 퍼블릭 DNS 이름 퍼블릭 DNS 이름은 EC2 콘솔에서 확인할 수 있습니다.
- 프라이빗 키 파일에 대한 키 페어 키 페어 액세스에 대한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서에서 [Amazon EC2 키 페어](#)를 참조하세요.

Windows 클라이언트에서 Jupyter Notebook 사용

Windows 클라이언트에서 Amazon EC2 인스턴스에 연결하는 방법은 아래의 가이드를 참조하세요.

1. [인스턴스 연결 문제 해결](#)
2. [PuTTY를 사용하여 Windows에서 Linux 인스턴스에 연결](#)

실행 중인 Jupyter 서버에 대한 터널을 만들려면 Windows 클라이언트에 Git Bash를 설치한 다음 Linux/macOS 클라이언트 지침을 따르는 것이 좋습니다. 그러나 포트 매핑을 사용하여 SSH 터널을 여는 데 필요한 모든 방법을 사용할 수 있습니다. 더 자세한 정보는 [Jupyter의 설명서](#)를 참조하십시오.

다음 단계

[Linux 또는 macOS 클라이언트 구성](#)

Linux 또는 macOS 클라이언트 구성

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 공

개 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

[Jupyter Notebook 서버에 로그인하여 설정 테스트](#)

Jupyter Notebook 서버에 로그인하여 설정 테스트

이제 Jupyter Notebook 서버에 로그인할 준비가 되었습니다.

다음 단계는 브라우저를 통해 서버 연결을 테스트하는 것입니다.

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.

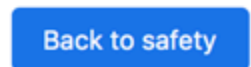


Your connection is not private

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

NET::ERR_CERT_AUTHORITY_INVALID

Help improve Safe Browsing by sending some [system information and page content](#) to Google.
[Privacy policy](#)



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



Your connection is not private

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

NET::ERR_CERT_AUTHORITY_INVALID

Help improve Safe Browsing by sending some [system information and page content](#) to Google.
[Privacy policy](#)

Hide advanced

Back to safety

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

[Proceed to localhost \(unsafe\)](#)

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

이제 DLMAI에서 실행 중인 Jupyter Notebook 서버에 액세스할 수 있습니다. 새 노트북을 생성하거나 [자습서](#)에서 제공된 노트북을 실행할 수 있습니다.

a 사용 DLAMI

주제

- [Conda를 통한 딥 러닝 사용 AMI](#)
- [딥러닝 베이스 사용 AMI](#)
- [Jupyter Notebook 자습서 실행](#)
- [자습서](#)

다음 섹션에서는 Conda를 AMI 통한 딥 러닝을 사용하여 환경을 전환하고, 각 프레임워크의 샘플 코드를 실행하고, Jupyter를 실행하여 다양한 노트북 자습서를 시험해 볼 수 있는 방법을 설명합니다.

Conda를 통한 딥 러닝 사용 AMI

주제

- [Conda를 AMI 사용한 딥 러닝 소개](#)
- [Your에 로그인하세요. DLAMI](#)
- [TensorFlow 환경을 시작하십시오.](#)
- [PyTorch Python 3 환경으로 전환하기](#)
- [환경 제거](#)

Conda를 AMI 사용한 딥 러닝 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 AMI 사용한 딥 러닝은 딥 러닝 환경 간에 쉽게 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. 그런 다음 각 프레임워크의 프로젝트 사이트에 있는 DLAMI 또는 프레임워크의 예제와 함께 제공되는 보다 자세한 자습서로 넘어갈 수 있습니다.

Your에 로그인하세요. DLAMI

서버에 로그인하면 다양한 딥 러닝 프레임워크 간에 전환하는 데 사용할 수 있는 다양한 Conda 명령을 설명하는 서버 “오늘의 메시지” (MOTD) 가 표시됩니다. 다음은 예제입니다. MOTD 새 버전이 출시됨에 따라 구체적인 내용이 MOTD 달라질 수 있습니다. DLAMI

```
=====
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 환경을 시작하십시오.

Note

Conda 환경을 처음 시작할 때 로드되는 동안 기다려야 합니다. Conda를 AMI 사용한 딥 러닝은 프레임워크가 처음 활성화될 때 EC2 인스턴스에 가장 최적화된 버전의 프레임워크를 자동으로 설치합니다. 이후에는 지연되지 않습니다.

1. Python 3용 TensorFlow 가상 환경을 활성화합니다.


```
$ source activate tensorflow2_p310
```

2. iPython 터미널을 시작합니다.

```
(tensorflow2_p310)$ ipython
```

3. 퀵 TensorFlow 프로그램을 실행하세요.

```
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 전환하기

아직 iPython 콘솔을 사용하고 있다면 를 사용하고 quit() 환경을 전환할 준비를 하세요.

- Python 3용 PyTorch 가상 환경을 활성화합니다.

```
$ source activate pytorch_p310
```

일부 PyTorch 코드 테스트

설치를 테스트하려면 Python을 사용하여 배열을 만들고 인쇄하는 PyTorch 코드를 작성하십시오.

1. iPython 터미널을 시작합니다.

```
(pytorch_p310)$ ipython
```

2. 가져오기 PyTorch.

```
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

의 공간이 부족한 경우 사용하지 않는 Conda 패키지를 제거하도록 선택할 수 있습니다. DLAMI

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

딥러닝 베이스 사용 AMI

딥러닝 베이스 사용 AMI

The AMI Base에는 자체 맞춤형 딥 러닝 환경을 배포할 수 있는 기본 GPU 드라이버 및 가속 라이브러리 플랫폼이 함께 제공됩니다. 기본적으로 AMI 는 단일 NVIDIA CUDA 버전 환경으로 구성됩니다. 의 다른 버전 간에 전환할 수도 CUDA 있습니다. 실행 방법은 다음 지침을 참조하세요.

CUDA버전 구성

NVIDIA의 `nvcc` 프로그램을 실행하여 CUDA 버전을 확인할 수 있습니다.

```
nvcc --version
```

다음 `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 노트](#)를 참조하십시오.

Jupyter Notebook 자습서 실행

자습서 및 예제는 각 딥 러닝 프로젝트의 소스와 함께 제공되며 대부분의 경우 어느 곳에서도 실행됩니다. DLAMI [AMIConda를 사용한 딥 러닝](#)(들) 선택했다면 설정 및 시험 사용이 준비된 일부 자습서로부터 추가 혜택을 받게 됩니다.

Important

에 설치된 Jupyter 노트북 자습서를 실행하려면 다음을 수행해야 합니다 DLAMI. [Jupyter Notebook 서버 설정](#)

Jupyter 서버가 실행 중이면 웹 브라우저를 통해 자습서를 실행할 수 있습니다. AMIConda로 딥 러닝을 실행하거나 Python 환경을 설정한 경우 Jupyter 노트북 인터페이스에서 Python 커널을 전환할 수 있습니다. 프레임워크별 자습서를 따라하기 전에 적절한 커널을 선택하세요. Conda를 사용한 딥 러닝 사용자 위해 이에 대한 추가 예제가 제공됩니다. AMI

Note

많은 자습서에는 사용자가 DLAMI 직접 설정하지 않을 수 있는 추가 Python 모듈이 필요합니다. 와 같은 "xyz module not found" 오류가 발생하면 에 로그인하여 위에서 설명한 대로 환경을 활성화한 다음 필요한 모듈을 설치하십시오. DLAMI

Tip

딥러닝 자습서 및 예제는 대개 하나 이상의 GPUs 내용을 기반으로 합니다. 인스턴스 유형에 GPU a가 없는 경우 예제 코드를 일부 변경하여 실행해야 할 수 있습니다.

설치된 자습서 탐색

Jupyter 서버에 로그인하고 튜토리얼 디렉터리 (Conda를 AMI 사용한 딥러닝에서만 해당) 를 볼 수 있게 되면 각 프레임워크 이름별로 튜토리얼 폴더가 표시됩니다. 프레임워크가 목록에 없다면 현재 사용 중인 프레임워크에서는 해당 프레임워크에 대한 자습서를 사용할 수 없는 것입니다. DLAMI 프레임워크의 이름을 클릭하면 자습서가 나열되고, 자습서를 클릭하면 이를 시작할 수 있습니다.

Conda를 AMI 사용하여 딥러닝으로 노트북을 처음 실행하면 어떤 환경을 사용하고 싶은지 알고 싶어질 것입니다. 선택할 목록이 표시됩니다. 각 환경은 이 패턴에 따라 이름이 지정됩니다.

Environment (conda_framework_python-version)

예를 들어, 환경에 MXNet Python 3이 있음을 나타내는 것을 볼 Environment (conda_mxnet_p36) 수 있습니다. 이것의 다른 변형은 다음과 같으며 Environment (conda_mxnet_p27), 이는 환경에 MXNet Python 2가 있음을 의미합니다.

Tip

어떤 버전이 활성화되어 있는지 걱정되는 경우 활성화된 버전을 확인할 수 있는 한 가지 방법은 처음 MOTD 로그인할 때 확인하는 것입니다. CUDA DLAMI

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. 여기에는 유용한 메시지와 오류 로깅이 있으며 out-of-memory 오류가 표시될 수 있습니다. 이를 수정하려면 Jupyter 서버의 홈 페이지로 이동하여 [Running] 탭을 클릭하고 여전히 백그라운드에서 실행

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

자습서

다음은 Conda AMI 소프트웨어와 함께 딥 러닝을 사용하는 방법에 대한 자습서입니다.

주제

- [프레임워크 활성화](#)
- [엘라스틱 패브릭 어댑터를 사용한 분산 교육](#)
- [GPU모니터링 및 최적화](#)
- [The AWS 인퍼런시아 칩 너비 DLAMI](#)
- [더 ARM64 DLAMI](#)
- [Inference](#)
- [모델 제공](#)

프레임워크 활성화

다음은 Conda를 AMI 사용한 딥 러닝에 설치된 딥 러닝 프레임워크입니다. 프레임워크를 클릭하면 활성화하는 방법을 자세히 알아볼 수 있습니다.

주제

- [PyTorch](#)
- [TensorFlow 2](#)

PyTorch

활성화하기 PyTorch

프레임워크의 안정적인 Conda 패키지가 출시되면 테스트를 거쳐 프레임워크에 사전 설치됩니다. DLAMI 테스트되지 않은 최신 야간 구축을 실행하려는 경우 수동으로 [야간 빌드 설치 PyTorch \(실험용\)](#)를 수행할 수 있습니다.

현재 설치된 프레임워크를 활성화하려면 Conda를 AMI 사용한 딥러닝의 다음 지침을 따르세요.

CUDA 및 PyTorch MKL -를 사용하는 Python DNN 3에서는 다음 명령을 실행하십시오.

```
$ source activate pytorch_p310
```

iPython 터미널을 시작합니다.

```
(pytorch_p310)$ ipython
```

퀵 PyTorch 프로그램을 실행하세요.

```
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 크기, 그리고 또 다른 임의 어레이가 추가된 것을 볼 수 있습니다.

야간 빌드 설치 PyTorch (실험용)

야간 PyTorch 빌드에서 설치하는 방법

Conda를 사용한 딥 AMI 러닝에서 PyTorch Conda 환경 중 하나 또는 둘 다에 최신 PyTorch 빌드를 설치할 수 있습니다.

- (파이썬 3을 위한 옵션) - 파이썬 3 PyTorch 환경을 활성화합니다:

```
$ source activate pytorch_p310
```

- 나머지 단계에서는 pytorch_p310 환경을 사용하고 있다고 가정합니다. 현재 설치된 파일 제거 PyTorch:

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

- (GPU인스턴스용 옵션) - 2.0을 PyTorch 사용하여 CUDA 최신 야간 빌드를 설치합니다.

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

- (CPU인스턴스용 옵션) - 다음을 사용하지 않고 인스턴스용 최신 야간 빌드를 설치합니다. PyTorch GPUs

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

4. 최신 야간 빌드를 성공적으로 설치했는지 확인하려면 IPython 터미널을 시작하고 버전을 확인하십시오. PyTorch

```
(pytorch_p310)$ ipython
```

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

출력은 1.0.0.dev20180922와 비슷하게 인쇄됩니다.

5. PyTorch 야간 빌드가 MNIST 예제와 잘 작동하는지 확인하려면 [examples](#) 저장소에서 PyTorch 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 자습서 및 예제는 프레임워크의 공식 [PyTorch 문서](#), [설명서](#) 및 웹 사이트를 참조하십시오.

[PyTorch](#)

TensorFlow 2

이 자습서에서는 Conda를 AMI 사용한 딥 러닝 (DLAMIConda) 을 실행하는 인스턴스에서 TensorFlow 2를 활성화하고 2 프로그램을 실행하는 방법을 보여줍니다. TensorFlow

프레임워크의 안정적인 Conda 패키지가 출시되면 테스트를 거쳐 프레임워크에 사전 설치됩니다.

DLAMI

2 활성화 중 TensorFlow

DLAMIConda와 함께 TensorFlow 실행하려면

1. TensorFlow 2를 활성화하려면 Conda를 사용하여 Amazon Elastic Compute Cloud (AmazonEC2) 인스턴스를 DLAMI 여십시오.

- Python 3에서 CUDA 10.1과 MKL -를 사용하는 2와 케라스 2의 DNN 경우 다음 명령을 실행합니다. TensorFlow

```
$ source activate tensorflow2_p310
```

- 터미널을 시작합니다. iPython

```
(tensorflow2_p310)$ ipython
```

- 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 자습서 및 예제는 [TensorFlow Python TensorFlow](#) 설명서 API 또는 [TensorFlow](#) 웹 사이트를 참조하십시오.

엘라스틱 패브릭 어댑터를 사용한 분산 교육

[Elastic Fabric Adapter](#) (EFA) 는 DLAMI 인스턴스에 연결하여 고성능 컴퓨팅 (HPC) 애플리케이션을 가속화할 수 있는 네트워크 장치입니다. EFA에서 제공하는 확장성, 유연성 및 탄력성을 통해 온프레미스 HPC 클러스터의 애플리케이션 성능을 달성할 수 있도록 합니다. AWS 클라우드.

다음 항목에서는 를 사용하여 EFA 시작하는 방법을 보여줍니다DLAMI.

Note

이 [기본 GPU DLAMI 목록에서](#) 원하는 DLAMI 것을 선택하세요.

주제

- [런칭: AWS Deep Learning AMIs 다음과 같은 인스턴스 EFA](#)

- [EFA에서 사용 DLAMI](#)

런칭: AWS Deep Learning AMIs 다음과 같은 인스턴스 EFA

최신 DLAMI 베이스는 필요한 드라이버, 커널 모듈, libfabric, openmpi EFA 및 예를 들어 [NCCLOFI 플러그인과](#) 함께 사용할 수 있으며 함께 제공됩니다. GPU

[릴리스 DLAMI 노트에서 지원되는 Base CUDA 버전을 찾을 수 있습니다.](#)

참고:

- mpirun을 사용하여 NCCL 애플리케이션을 실행하는 EFA 경우 EFA 지원되는 설치의 전체 경로를 다음과 같이 지정해야 합니다.

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

- 애플리케이션이 사용할 EFA 수 있게 FI_PROVIDER="efa" 하려면 와 같이 mpirun 명령에 추가하십시오 [EFA에서 사용 DLAMI](#).

주제

- [EFA를 사용한 보안 그룹 준비](#)
- [인스턴스 시작](#)
- [EFA 첨부 파일 확인](#)

EFA를 사용한 보안 그룹 준비

EFA보안 그룹 자체에서 들어오고 나가는 모든 인바운드 및 아웃바운드 트래픽을 허용하는 보안 그룹이 필요합니다. [자세한 내용은 설명서를 참조하십시오. EFA](#)

1. 에서 Amazon EC2 콘솔을 엽니다 <https://console.aws.amazon.com/ec2/>.
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 group과 같은 보안 그룹의 고유한 이름을 입력합니다.
 - (선택 사항) 설명에 보안 그룹에 대한 간략한 설명을 입력합니다.
 - 에서 VPC EFA -enabled 인스턴스를 시작하려는 인스턴스를 선택합니다. VPC

- 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.
 - 저장(Save)을 선택합니다.
 6. [Linux 인스턴스에 대한 인바운드 트래픽 권한 부여](#)를 참조하여 인바운드 트래픽을 활성화합니다. 이 단계를 건너뛰면 인스턴스와 통신할 수 없습니다. DLAMI

인스턴스 시작

EFA에서 AWS Deep Learning AMIs 현재 지원되는 인스턴스 유형 및 운영 체제는 다음과 같습니다.

- P3dn.24xlarge: 아마존 리눅스 2, 우분투 20.04
- P4D.24xlarge: 아마존 리눅스 2, 우분투 20.04
- P5.48xlarge: 아마존 리눅스 2, 우분투 20.04

다음 섹션에서는 활성화된 인스턴스를 시작하는 방법을 보여줍니다. EFA DLAMI EFA활성화된 인스턴스를 시작하는 방법에 대한 자세한 내용은 [클러스터 배치 그룹으로 EFA -Enabled 인스턴스 시작](#)을 참조하십시오.

1. 에서 Amazon EC2 콘솔을 엽니다 <https://console.aws.amazon.com/ec2/>.
2. 인스턴스 시작을 선택합니다.
3. AMI페이지 선택에서 DLAMI [릴리스 노트 페이지](#)에 있는 지원 DLAMI 항목을 선택합니다.
4. 인스턴스 유형 선택 페이지에서 다음과 같이 지원되는 인스턴스 유형 중 하나를 선택하고 다음: 인스턴스 세부 정보 구성을 선택합니다. 지원되는 인스턴스 목록은 이 링크를 참조하십시오. [시작하기 EFA 및 MPI](#)
5. 인스턴스 세부 정보 구성 페이지에서 다음을 수행합니다.
 - 인스턴스 수에는 시작하려는 EFA -enabled 인스턴스 수를 입력합니다.
 - 네트워크 및 서브넷의 경우 인스턴스를 시작할 VPC 서브넷과 서브넷을 선택합니다.

- [선택 사항] 배치 그룹의 경우 배치 그룹에 인스턴스 추가를 선택합니다. 최상의 성능을 위해 배치 그룹 내에서 인스턴스를 시작합니다.
 - [선택 사항] 배치 그룹 이름으로 새 배치 그룹에 추가를 선택하고 배치 그룹을 설명하는 이름을 입력한 다음 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. 선택적으로 기본 IPv4 주소와 하나 이상의 보조 IPv4 주소를 지정할 수 있습니다. 연결된 IPv6 CIDR 블록이 있는 서브넷에서 인스턴스를 시작하는 경우 선택적으로 기본 IPv6 주소와 하나 이상의 보조 IPv6 주소를 지정할 수 있습니다.
 - 다음: 스토리지 추가를 선택합니다.
6. Add Storage 페이지에서 에서 지정한 볼륨 AMI (예: 루트 디바이스 볼륨) 외에 인스턴스에 연결할 볼륨을 지정하고 [다음: Add Tags] 를 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 선택한 다음 이전에 만든 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 첨부 파일 확인

콘솔에서

인스턴스를 시작한 후에는 다음에서 인스턴스 세부 정보를 확인하십시오. AWS 콘솔. 이렇게 하려면 EC2 콘솔에서 인스턴스를 선택하고 페이지 하단 창의 Description 탭을 확인합니다. 'Network Interfaces: eth0' 매개 변수를 찾아 eth0을 클릭하면 팝업이 열립니다. 'Elastic Fabric Adapter'가 활성화되어 있는지 확인합니다.

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

- 동일한 단계를 사용하여 EC2 인스턴스를 종료하고 새 인스턴스를 시작합니다. EFA이 연결되어 있는지 확인합니다.
- 기존 인스턴스에 EFA를 연결합니다.

1. EC2콘솔에서 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. EC2인스턴스 탭으로 돌아가서 인스턴스를 선택합니다.
6. Actions: Instance State로 이동하여 연결하기 전에 인스턴스를 중지하십시오EFA.
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 스크립트는 이미 에 DLAMI 있습니다. 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 스크립트는 에 이미 DLAMI 있습니다. 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.

EFA에서 사용 DLAMI

다음 섹션에서는 에서 다중 노드 애플리케이션을 실행하는 EFA 데 사용하는 방법을 설명합니다. AWS Deep Learning AMIs.

EFA을 사용하여 다중 노드 애플리케이션 실행

노드 클러스터에서 애플리케이션을 실행하려면 다음과 같은 구성이 필요합니다.

주제

- [비밀번호 없음 활성화 SSH](#)
- [호스트 파일 생성](#)
- [NCCL테스트](#)

비밀번호 없음 활성화 SSH

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

1. 리더 노드에서 키페어를 생성합니다. RSA

```
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 클러스터의 멤버 ~/.ssh/id_rsa.pub 노드에 복사하고 ~/.ssh/authorized_keys 클러스터의 멤버 노드에 추가합니다.
4. 이제 프라이빗 IP를 사용하여 리더 노드에서 멤버 노드에 직접 로그인 할 수 있습니다.

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

5. 리더 노드의 ~/.ssh/config 파일에 다음을 추가하여 strictHostKey 검사를 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
```

NCCL테스트

Note

이 테스트는 EFA 버전 1.30.0 및 OFI NCCL 플러그인 1.7.4를 사용하여 실행되었습니다.

다음은 여러 컴퓨팅 노드에서 기능과 성능을 모두 NCCL 테스트하기 위해 Nvidia에서 제공하는 테스트의 하위 집합입니다.

지원되는 인스턴스: P3dn, P4, P5

기능 테스트

NCCL메시지 전송 멀티노드 테스트

nccl_message_transfer는 플러그인이 예상대로 작동하는지 확인하기 위한 간단한 테스트입니다. NCCL OFI 이 테스트는 의 연결 설정 및 데이터 전송 기능을 검증합니다. NCCL APIs 를 사용하여 애플리케이션을 실행하는 NCCL 동안 예제에 표시된 대로 mpirun의 전체 경로를 사용해야 합니다. EFA 인스턴스 수와 클러스터에 있는 인스턴스에 N 따라 매개변수를 np 변경하십시오GPU. 자세한 내용은 [단원을 참조하십시오.AWS OFINCCCL설명서](#).

다음 nccl_message_transfer 테스트는 일반 xx.x 버전을 대상으로 합니다. CUDA 스크립트에서 CUDA 버전을 교체하여 Amazon EC2 인스턴스에서 사용 가능한 모든 CUDA 버전의 명령을 실행할 수 있습니다.

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

출력은 다음과 같아야 합니다. 출력을 확인하여 OFI 공급자로 EFA 사용되고 있는 것을 확인할 수 있습니다.

```
INFO: Function: nccl_net_ofi_init Line: 1069: NET/OFI Selected Provider is efa (found 4  
 nics)
```

```

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

```

성능 테스트

P4D.24xlarge에서의 다중 노드 NCCL 성능 테스트

NCCL 성능을 확인하려면 공식 [NCCL-Tests 리포지토리](#)에서 제공되는 [표준 NCCL 성능 테스트](#)를 실행하세요. EFA는 이미 CUDA XXX용으로 빌드된 이 테스트와 함께 DLAMI 제공됩니다. 비슷한 방법으로 자체 스크립트를 실행할 수 있습니다. EFA

자체 스크립트를 작성할 때 다음 지침을 참조하세요.

- 로 애플리케이션을 실행하는 동안 예제에 표시된 대로 mpirun의 전체 경로를 사용하십시오. NCCL EFA
- 인스턴스 수와 클러스터에 있는 매개 변수 np와 N을 변경합니다. GPUs
- NCCL_DEBUG = INFO 플래그를 추가하고 로그에 EFA 사용량이 "Selected Provider isEFA"로 표시되는지 확인합니다.
- 검증을 위해 파싱할 트레이닝 로그 위치를 설정합니다.

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


모든 멤버 `watch nvidia-smi` 노드에서 명령을 사용하여 GPU 사용량을 모니터링할 수 있습니다. 다음 `watch nvidia-smi` 명령은 일반 CUDA `xx.x` 버전용이며 인스턴스의 운영 체제에 따라 다릅니다. 스크립트에서 CUDA 버전을 교체하여 Amazon EC2 인스턴스에서 사용 가능한 모든 CUDA 버전의 명령을 실행할 수 있습니다.

- Amazon Linux 2:

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

- 우분투 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 9591 on ip-172-31-4-37 device 0 [0x10] NVIDIA A100-SXM4-40GB
# Rank 1 Group 0 Pid 9592 on ip-172-31-4-37 device 1 [0x10] NVIDIA A100-SXM4-40GB
# Rank 2 Group 0 Pid 9593 on ip-172-31-4-37 device 2 [0x20] NVIDIA A100-SXM4-40GB
# Rank 3 Group 0 Pid 9594 on ip-172-31-4-37 device 3 [0x20] NVIDIA A100-SXM4-40GB
# Rank 4 Group 0 Pid 9595 on ip-172-31-4-37 device 4 [0x90] NVIDIA A100-SXM4-40GB
# Rank 5 Group 0 Pid 9596 on ip-172-31-4-37 device 5 [0x90] NVIDIA A100-SXM4-40GB
```

```
# Rank 6 Group 0 Pid 9597 on ip-172-31-4-37 device 6 [0xa0] NVIDIA A100-SXM4-40GB
# Rank 7 Group 0 Pid 9598 on ip-172-31-4-37 device 7 [0xa0] NVIDIA A100-SXM4-40GB
# Rank 8 Group 0 Pid 10216 on ip-172-31-13-179 device 0 [0x10] NVIDIA A100-
SXM4-40GB
# Rank 9 Group 0 Pid 10217 on ip-172-31-13-179 device 1 [0x10] NVIDIA A100-
SXM4-40GB
# Rank 10 Group 0 Pid 10218 on ip-172-31-13-179 device 2 [0x20] NVIDIA A100-
SXM4-40GB
# Rank 11 Group 0 Pid 10219 on ip-172-31-13-179 device 3 [0x20] NVIDIA A100-
SXM4-40GB
# Rank 12 Group 0 Pid 10220 on ip-172-31-13-179 device 4 [0x90] NVIDIA A100-
SXM4-40GB
# Rank 13 Group 0 Pid 10221 on ip-172-31-13-179 device 5 [0x90] NVIDIA A100-
SXM4-40GB
# Rank 14 Group 0 Pid 10222 on ip-172-31-13-179 device 6 [0xa0] NVIDIA A100-
SXM4-40GB
# Rank 15 Group 0 Pid 10223 on ip-172-31-13-179 device 7 [0xa0] NVIDIA A100-
SXM4-40GB
ip-172-31-4-37:9591:9591 [0] NCCL INFO Bootstrap : Using ens32:172.31.4.37
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin_v6
symbol.
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin
symbol (v4 or v5).
ip-172-31-4-37:9591:9591 [0] NCCL INFO cudaDriverVersion 12020
NCCL version 2.18.5+cuda12.2
...
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.7.4-aws
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
variable to 1
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
variable to 1
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Running on p4d.24xlarge platform,
Setting NCCL_TOPO_FILE environment variable to /opt/aws-ofi-nccl/share/aws-ofi-nccl/
xml/p4d-24x1-topo.xml
```

```

...
-----some output truncated-----
#
#           in-place                                     out-of-place
#   size      count   type  redop  root   time  algbw  busbw #wrong
#   time  algbw  busbw #wrong
#   (us)  (GB/s) (GB/s)
#           0      0   float  sum    -1   11.02  0.00  0.00  0
11.04  0.00  0.00  0
#           0      0   float  sum    -1   11.01  0.00  0.00  0
11.00  0.00  0.00  0
#           0      0   float  sum    -1   11.02  0.00  0.00  0
11.02  0.00  0.00  0
#           0      0   float  sum    -1   11.01  0.00  0.00  0
11.00  0.00  0.00  0
#           0      0   float  sum    -1   11.02  0.00  0.00  0
11.02  0.00  0.00  0
#           256     4   float  sum    -1   632.7  0.00  0.00  0
628.2  0.00  0.00  0
#           512     8   float  sum    -1   627.4  0.00  0.00  0
629.6  0.00  0.00  0
#           1024    16  float  sum    -1   632.2  0.00  0.00  0
631.7  0.00  0.00  0
#           2048    32  float  sum    -1   631.0  0.00  0.00  0
634.2  0.00  0.00  0
#           4096    64  float  sum    -1   623.3  0.01  0.01  0
633.6  0.01  0.01  0
#           8192   128  float  sum    -1   635.1  0.01  0.01  0
633.5  0.01  0.01  0
#           16384   256  float  sum    -1   634.8  0.03  0.02  0
637.0  0.03  0.02  0
#           32768   512  float  sum    -1   647.9  0.05  0.05  0
636.8  0.05  0.05  0
#           65536  1024  float  sum    -1   658.9  0.10  0.09  0
667.0  0.10  0.09  0
#           131072  2048  float  sum    -1   671.9  0.20  0.18  0
662.9  0.20  0.19  0
#           262144  4096  float  sum    -1   692.1  0.38  0.36  0
685.1  0.38  0.36  0
#           524288  8192  float  sum    -1   715.3  0.73  0.69  0
696.6  0.75  0.71  0
#           1048576 16384  float  sum    -1   734.6  1.43  1.34  0
729.2  1.44  1.35  0

```

```

    2097152      32768      float      sum      -1      785.9      2.67      2.50      0
794.5    2.64    2.47    0
    4194304      65536      float      sum      -1      837.2      5.01      4.70      0
837.6    5.01    4.69    0
    8388608      131072     float      sum      -1      929.2      9.03      8.46      0
931.4    9.01    8.44    0
    16777216     262144     float      sum      -1      1773.6     9.46      8.87      0
1772.8   9.46    8.87    0
    33554432     524288     float      sum      -1      2110.2     15.90     14.91     0
2116.1   15.86   14.87    0
    67108864     1048576    float      sum      -1      2650.9     25.32     23.73     0
2658.1   25.25   23.67    0
    134217728    2097152    float      sum      -1      3943.1     34.04     31.91     0
3945.9   34.01   31.89    0
    268435456    4194304    float      sum      -1      7216.5     37.20     34.87     0
7178.6   37.39   35.06    0
    536870912    8388608    float      sum      -1      13680      39.24     36.79     0
13676    39.26   36.80    0
[ 1073741824    16777216    float      sum      -1      25645      41.87     39.25     0
25497    42.11   39.48    0 ] <- Used For Benchmark
...
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 7.46044

```

검증 테스트

EFA테스트 결과가 유효한지 확인하려면 다음 테스트를 사용하여 확인하십시오.

- 인스턴스 메타데이터를 사용하여 EC2 인스턴스 유형을 가져오십시오.

```

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)

```

- 를 실행합니다. [성능 테스트](#)
- 다음 파라미터를 설정합니다.

```

CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION

```

- 다음과 같이 결과를 검증합니다.

```

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

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

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

    # Validation of logs
    grep "NET/OFI Using CUDA runtime version ${CUDA_RUNTIME_VERSION}" ${TRAINING_LOG}
    || { echo "Runtime cuda text not found"; exit 1; }
    grep "NET/OFI Initializing aws-ofi-nccl" ${TRAINING_LOG} || { echo "aws-ofi-nccl
    is not working, please check if it is installed correctly"; exit 1; }
    grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-
    specific options text not found"; exit 1; }
    grep "Using network AWS Libfabric" ${TRAINING_LOG} || { echo "AWS Libfabric text
    not found"; exit 1; }
    grep "busbw" ${TRAINING_LOG} || { echo "busbw text not found"; exit 1; }
    grep "Avg bus bandwidth " ${TRAINING_LOG} || { echo "Avg bus bandwidth text not
    found"; exit 1; }
    grep "NCCL version $NCCL_VERSION" ${TRAINING_LOG} || { echo "Text not found: NCCL
    version $NCCL_VERSION"; exit 1; }

    if [[ ${INSTANCE_TYPE} == "p4d.24xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found:
    NET/AWS Libfabric/0/GDRDMA"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
    { echo "Selected Provider is efa text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p4d-24xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
    file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
    bandwidth text not found"; exit 1; }

```

```

    grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
{ echo "Avg bus bandwidth text not found"; exit 1; }
    grep "aws-ofi-nccl/xml/p4de-24x1-topo.xml" ${TRAINING_LOG} || { echo
"Topology file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
bandwidth text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
{ echo "Avg bus bandwidth text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p5.48x1-topo.xml" ${TRAINING_LOG} || { echo "Topology
file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p3dn.24xlarge" ]]; then
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
{ echo "Selected Provider is efa text not found"; exit 1; }
    fi
    echo "***** check_efa_nccl_all_reduce passed for cuda
version ${CUDA_VERSION} *****"
else
    echo "***** check_efa_nccl_all_reduce failed for cuda
version ${CUDA_VERSION} *****"
fi

```

- 벤치마크 데이터에 액세스하기 위해 Multi Node all_reduce 테스트의 테이블 출력의 마지막 행을 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}"

```

GPU모니터링 및 최적화

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

- [모니터링](#)

- [GPUs다음을 사용하여 모니터링하십시오. CloudWatch](#)
- [최적화](#)
- [사전 처리](#)
- [학습](#)

모니터링

몇 가지 GPU 모니터링 DLAMI 도구가 사전 설치되어 있습니다. 이 설명서에서는 다운로드 및 설치에 사용할 수 있는 도구에 대해서도 설명합니다.

- [GPUs다음을 사용하여 모니터링하십시오. CloudWatch](#) - CloudWatch Amazon에 GPU 사용 통계를 보고하는 사전 설치된 유틸리티입니다.
- [nvidia-smi CLI](#) - 전체 컴퓨팅 및 메모리 사용률을 모니터링하는 유틸리티입니다. GPU 이 앱은 컴퓨터에 사전 설치되어 있습니다. AWS Deep Learning AMLs (DLAMI).
- [NVMLC 라이브러리](#) - GPU 모니터링 및 관리 기능에 직접 액세스할 API 수 있는 C 기반 라이브러리입니다. 이는 CLI nvidia-smi에서 기본적으로 사용되며 사용자 컴퓨터에 사전 설치되어 있습니다. DLAMI 또한 Python 및 Perl 바인딩을 가지고 있어 이러한 언어로 신속하게 개발이 가능합니다. 사용자 컴퓨터에 사전 설치된 gpumon.py 유틸리티는 의 pynvml 패키지를 사용합니다. DLAMI [nvidia-ml-py](#)
- [NVIDIADCGM](#)- 클러스터 관리 도구. 이 도구를 설치하고 구성하는 방법을 알아보려면 개발자 페이지를 방문하세요.

Tip

설치된 CUDA 도구 사용에 대한 최신 정보는 NVIDIA 의 개발자 블로그에서 확인하세요 DLAMI.

- [IDENsight와 nvprof를 사용하여 TensorCore 사용률을 모니터링합니다.](#)

GPUs다음을 사용하여 모니터링하십시오. CloudWatch

DLAMIa와 함께 사용하는 GPU 경우 훈련 또는 추론 중에 사용량을 추적할 방법을 찾고 있을 수 있습니다. 이는 데이터 파이프라인을 최적화하고 딥 러닝 네트워크를 튜닝할 때 유용할 수 있습니다.

다음과 같은 두 가지 방법으로 GPU 메트릭을 CloudWatch 구성할 수 있습니다.

- [다음](#)을 사용하여 메트릭을 구성합니다. [AWS CloudWatch 에이전트 \(권장\)](#)
- [사전 설치된 gpumon.py 스크립트로 지표를 구성합니다.](#)

다음을 사용하여 메트릭을 구성합니다. [AWS CloudWatch 에이전트 \(권장\)](#)

[통합 CloudWatch 에이전트와](#) 통합하여 Amazon EC2 액셀러레이티드 인스턴스에서 GPU 지표를 구성하고 GPU 코프로세스 사용률을 모니터링할 수 있습니다. DLAMI

다음을 사용하여 [GPU메트릭](#)을 구성하는 네 가지 방법이 있습니다DLAMI.

- [최소 GPU 지표 구성](#)
- [부분 GPU 지표 구성](#)
- [사용 가능한 GPU 모든 지표를 구성합니다.](#)
- [사용자 지정 GPU 메트릭을 구성합니다.](#)

업데이트 및 보안 패치에 관한 자세한 내용은 [에 대한 보안 패치 AWS CloudWatch 에이전트](#)를 참조하세요.

사전 조건

시작하려면 인스턴스가 메트릭을 푸시할 수 있도록 Amazon EC2 인스턴스 IAM 권한을 구성해야 CloudWatch 합니다. 자세한 단계는 [CloudWatch 에이전트와 함께 사용할 IAM 역할 및 사용자 생성](#)을 참조하십시오.

최소 GPU 지표 구성

dlami-cloudwatch-agent@minimalsystemd서비스를 사용하여 최소 GPU 지표를 구성합니다. 이 서비스의 지표 구성은 다음과 같습니다.

- utilization_gpu
- utilization_memory

사전 구성된 최소 GPU 지표에 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
```

부분 GPU 지표 구성

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

- utilization_gpu
- utilization_memory
- memory_total
- memory_used
- memory_free

사전 구성된 일부 GPU 지표에 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 모든 지표를 구성합니다.

dlami-cloudwatch-agent@allsystemd 서비스를 사용하여 사용 가능한 모든 GPU 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`

사용 가능한 모든 사전 구성 GPU 지표에 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 메트릭을 구성합니다.

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

사용자 지정 구성 파일 생성

사용자 지정 구성 파일을 생성하려면 [CloudWatch 에이전트 구성 파일 수동 생성 또는 편집의](#) 세부 단계를 참조하십시오.

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
```

에 대한 보안 패치 AWS CloudWatch 에이전트

새로 DLAMIs 릴리스된 버전은 사용 가능한 최신 버전으로 구성되어 있습니다. AWS CloudWatch 에이전트 보안 패치. 선택한 운영 체제에 따라 최신 보안 DLAMI 패치로 최신 보안 패치를 업데이트하려면 다음 섹션을 참조하십시오.

Amazon Linux 2

최신 버전을 yum 다운로드하는 데 사용합니다. AWS CloudWatch Amazon Linux DLAMI 2용 에이전트 보안 패치

```
sudo yum update
```

Ubuntu

최신 정보를 얻으려면 AWS CloudWatch DLAMIUbuntu를 사용하는 a용 보안 패치를 다시 설치해야 합니다. AWS CloudWatch Amazon S3 다운로드 링크를 사용하는 에이전트

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

설치에 대한 자세한 내용은 AWS CloudWatch Amazon S3 다운로드 링크를 사용하는 [CloudWatch 에이전트는 서버에 에이전트 설치 및 실행을](#) 참조하십시오.

사전 설치된 **gpumon.py** 스크립트로 지표를 구성합니다.

gpumon.py 라는 유틸리티가 사전 설치되어 있습니다. DLAMI GPU메모리, GPU 온도 CloudWatch 및 전력과 통합되어 GPU 사용량별 모니터링을 지원합니다. GPU 스크립트는 모니터링된 데이터를 주기적으로 CloudWatch 전송합니다. 스크립트에서 몇 가지 설정을 CloudWatch 변경하여 전송 대상 데이터의 세분화 수준을 구성할 수 있습니다. 하지만 스크립트를 시작하기 전에 메트릭을 CloudWatch 수신하도록 설정해야 합니다.

를 사용하여 GPU 모니터링을 설정하고 실행하는 방법 CloudWatch

1. 지표를 게시하기 위한 정책을 갖도록 IAM 사용자를 만들거나 기존 사용자를 CloudWatch 수정하십시오. 새로 사용자를 생성하는 경우에는 다음 단계에서 필요할 수 있기 때문에 자격 증명을 적어 두십시오.

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

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

Tip

[IAM 사용자 생성 및 정책 추가에 대한 CloudWatch 자세한 내용은 설명서를 CloudWatch 참조하십시오.](#)

- 에서 DLAMI 다음을 실행하십시오. [AWS IAM 사용자 자격 증명을](#) 구성하고 지정합니다.

```
$ aws configure
```

- 실행에 앞서 gpumon 유틸리티를 약간 수정해야 할 수 있습니다. gpumon 유틸리티는 다음 코드 README 블록에 정의된 위치에서 찾을 수 있습니다. gpumon.py 스크립트에 대한 자세한 내용은 [스크립트의 Amazon S3 위치](#)를 참조하세요.

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

옵션:

- 인스턴스가 us-east-1에 있는 경우 gpumon.py NOT 에서 지역을 변경하십시오.
 - 를 사용하여 CloudWatch namespace 또는 보고 기간과 같은 기타 파라미터를 변경하십시오.
store_reso
- 현재, 스크립트는 Python 3만 지원합니다. 선호하는 프레임워크의 Python 3 환경을 활성화하거나 DLAMI 일반 Python 3 환경을 활성화합니다.

```
$ source activate python3
```

5. 배경에서 gpumon 유틸리티를 실행합니다.

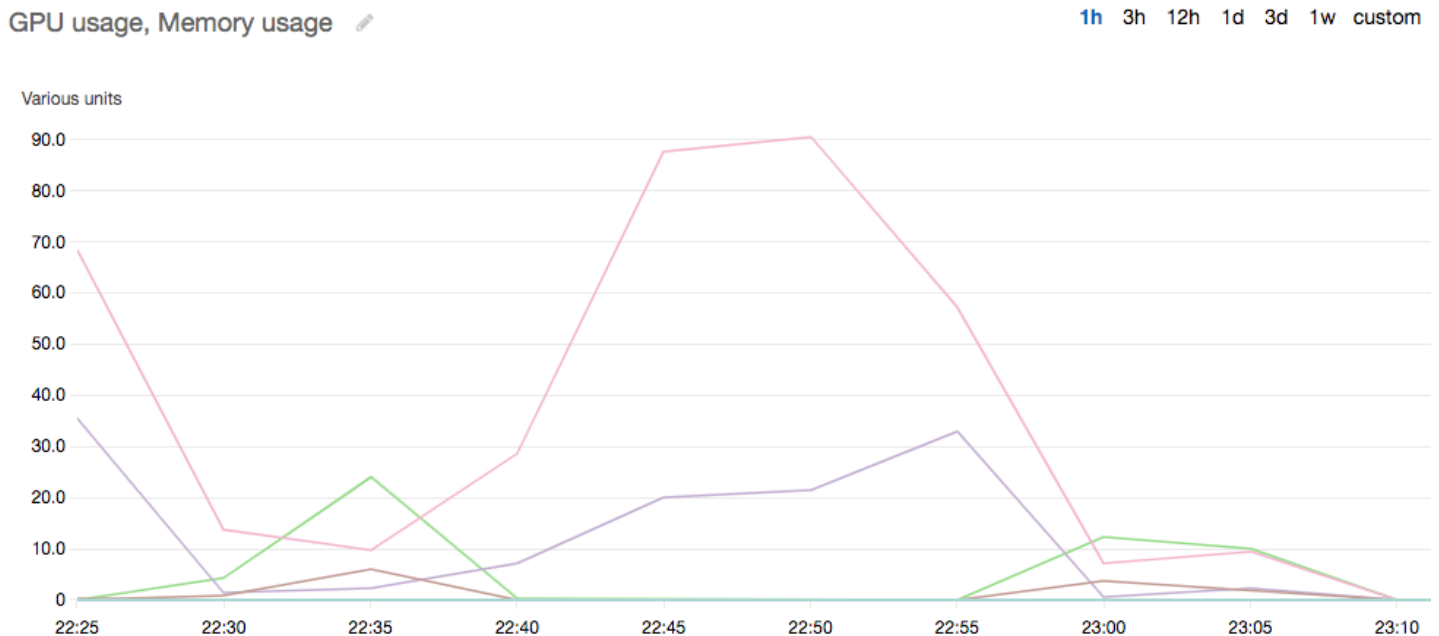
```
(python3)$ python gpumon.py &
```

6. 브라우저를 열고 메트릭을 <https://console.aws.amazon.com/cloudwatch/> 선택합니다. 네임스페이스는 'DeepLearningTrain'입니다.

Tip

gpumon.py를 수정하여 네임스페이스를 변경할 수 있습니다. 또한 store_reso를 조정하여 보고 간격을 수정할 수도 있습니다.

다음은 CloudWatch p2.8xlarge 인스턴스에서 교육 작업을 모니터링하는 gpumon.py 실행에 대한 예제 차트입니다.



GPU모니터링 및 최적화에 관한 다음과 같은 다른 항목에도 관심이 있을 수 있습니다.

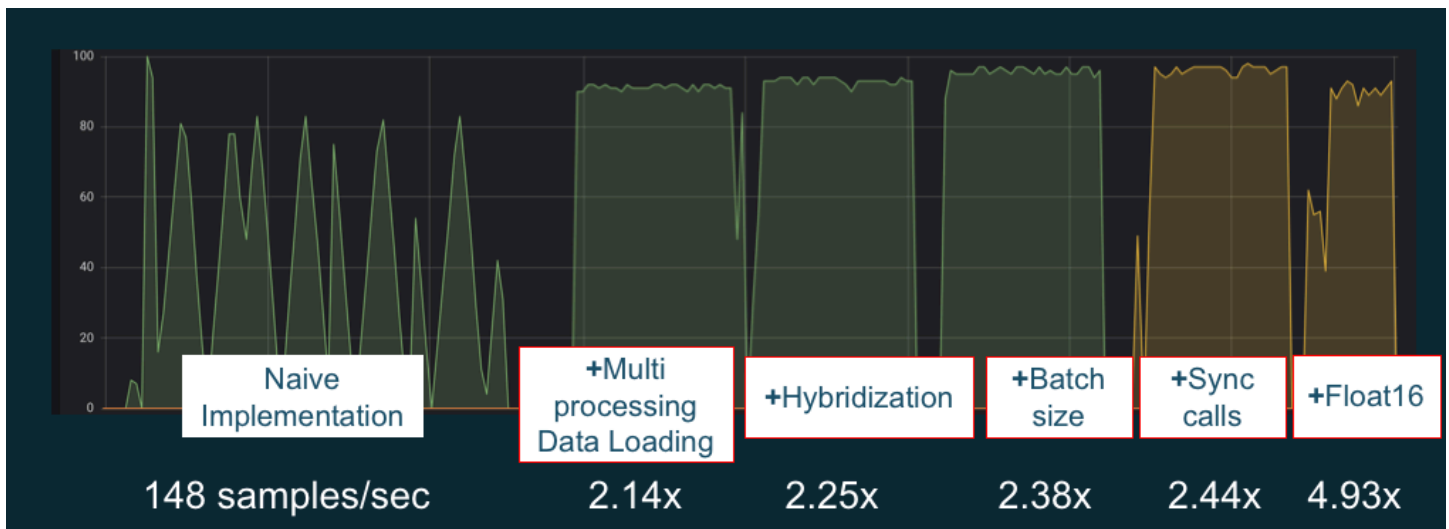
- [모니터링](#)
 - [GPUs다음을 사용하여 모니터링하십시오. CloudWatch](#)
- [최적화](#)

- [사전 처리](#)
- [학습](#)

최적화

이를 최대한 활용하기 위해 데이터 파이프라인을 최적화하고 딥러닝 네트워크를 조정할 수 있습니다. GPUs 다음 차트에서 설명하는 것처럼, 순진하거나 기본적인 신경망 구현은 GPU 일관되지 않은 요소를 사용하고 잠재력을 최대한 발휘하지 못할 수 있습니다. 전처리 및 데이터 로딩을 최적화하면 작업에서 발생한 병목 현상을 줄일 수 있습니다. CPU GPU 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.



GPU성능 최적화의 예. 차트 출처: [Glueon을 사용한 퍼포먼스 트릭 MXNet](#)

다음 가이드에서는 사용자에게 DLAMI 적합하고 GPU 성능을 높이는 데 도움이 되는 옵션을 소개합니다.

주제

- [사전 처리](#)
- [학습](#)

사전 처리

변환이나 확장을 통한 데이터 전처리는 종종 제한된 프로세스일 수 있으며, 이는 전체 CPU 파이프라인에서 병목 현상이 발생할 수 있습니다. 프레임워크에는 이미지 처리를 위한 내장 연산자가 있지만 DALI (데이터 증강 라이브러리) 는 프레임워크의 기본 제공 옵션에 비해 성능이 향상되었음을 보여줍니다.

- NVIDIA데이터 증강 라이브러리 (DALI): DALI 데이터 증강을 로 오프로드합니다. GPU 에는 사전 설치되어 있지 않지만 사용자 또는 다른 Amazon Elastic Compute Cloud 인스턴스에 지원되는 프레임워크 컨테이너를 DLAMI 설치하거나 로드하여 액세스할 수 있습니다. DLAMI 자세한 내용은 NVIDIA 웹 사이트의 [DALI프로젝트 페이지](#)를 참조하십시오. 예제 사용 사례와 코드 샘플 다운로드에는 [SageMaker 사전 처리 교육](#) 성능 샘플을 참조하십시오.
- nvJPEG: C 프로그래머를 위한 GPU 가속 JPEG 디코더 라이브러리. [단일 이미지 또는 배치 디코딩은 물론 딥러닝에서 흔히 사용되는 후속 변환 작업도 지원합니다.](#) nv는 기본 JPEG 제공되거나 웹 사이트의 [nvjpeg 페이지](#)에서 다운로드하여 별도로 사용할 수 있습니다. [DALI NVIDIA](#)

모니터링 및 최적화에 관한 다음과 같은 다른 주제에도 관심이 있을 수 있습니다. GPU

- [모니터링](#)
 - [GPUs다음을 사용하여 모니터링하십시오. CloudWatch](#)
- [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.

- [NVIDIA딥러닝 SDK](#) -, 및 에 대한 혼합 정밀도 구현을 설명하는 NVIDIA 웹 사이트 문서 MXNet PyTorch TensorFlow

i 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.

- [TensorFlow \(비디오\) 를 통한 혼합 정밀도 교육](#) - 블로그 사이트에서 NVIDIA
- [float16을 사용한 혼합 정밀도 훈련 MXNet - 웹 사이트에 실린 기사 FAQ MXNet](#)
- [NVIDIAApex: 간편한 혼합 정밀 교육을 위한 도구 PyTorch - 웹 사이트의 블로그 기사.](#)
NVIDIA

GPU모니터링 및 최적화에 관한 다음과 같은 다른 주제에도 관심이 있을 수 있습니다.

- [모니터링](#)
 - [GPUs다음을 사용하여 모니터링하십시오. CloudWatch](#)
- [최적화](#)
 - [사전 처리](#)
 - [학습](#)

The AWS 인퍼런시아 칩 너비 DLAMI

AWS Inferentia는 다음과 같이 설계된 맞춤형 기계 학습 칩입니다. AWS 고성능 추론 예측에 사용할 수 있습니다. 이 칩을 사용하려면 Amazon Elastic Compute Cloud 인스턴스를 설정하고 다음을 사용하십시오. AWS 뉴런 소프트웨어 개발 키트 (SDK) 를 사용하여 Inferentia 칩을 호출합니다. 고객에게 최고의 Inferentia 경험을 제공하기 위해 Neuron이 내장되어 있습니다. AWS Deep Learning AMIs (DLAMI).

다음 항목에서는 Inferentia를 사용하여 시작하는 방법을 보여줍니다. DLAMI

내용

- [를 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)
- [with 사용 DLAMI AWS Neuron](#)

를 사용하여 DLAMI 인스턴스 시작 AWS Neuron

최신 DLAMI 버전은 다음과 함께 사용할 수 있습니다. AWS Inferentia 및 함께 제공됩니다. AWS 뉴런 패키지API. DLAMI인스턴스를 시작하려면 [시작 및 구성을 참조하십시오. DLAMI](#) 실행한 DLAMI 후에

는 여기의 단계를 사용하여 다음을 확인하십시오. AWS 인퍼런시아 칩 및 AWS 뉴런 리소스가 활성화 상태입니다.

내용

- [인스턴스 확인](#)
- [식별 AWS 추론 디바이스](#)
- [리소스 사용량 보기](#)
- [Neuron Monitor 사용\(neuron-monitor\)](#)
- [Neuron 소프트웨어 업그레이드](#)

인스턴스 확인

인스턴스를 사용하기 전에 인스턴스가 제대로 설정되고 Neuron으로 구성되어 있는지 확인합니다.

식별 AWS 추론 디바이스

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.

- NEURONDEVICE: 에 할당된 논리적 ID입니다. NeuronDevice 이 ID는 여러 런타임에서 서로 다른 NeuronDevices 것을 사용하도록 구성할 때 사용됩니다.
- NEURONCORES: 에 NeuronCores NeuronDevice 존재하는 개수.
- NEURONMEMORY: 에 있는 DRAM 메모리의 양 NeuronDevice.

- CONNECTEDDEVICES: 기타가 에 NeuronDevices 연결되어 NeuronDevice 있습니다.
- PCIBDF: 의 PCI 버스 디바이스 함수 (BDF) ID NeuronDevice.

리소스 사용량 보기

neuron-top 명령을 사용하여 NeuronCore 및 v CPU 사용률, 메모리 사용량, 로드된 모델 및 Neuron 애플리케이션에 대한 유용한 정보를 볼 수 있습니다. 인수 없이 실행하면 neuron-top 이를 활용하는 NeuronCores 모든 기계 학습 응용 프로그램의 데이터가 표시됩니다.

```
neuron-top
```

애플리케이션이 4개를 사용하는 NeuronCores 경우 출력은 다음 이미지와 비슷해야 합니다.



Neuron 기반 추론 애플리케이션을 모니터링하고 최적화하기 위한 리소스에 대한 자세한 내용은 [Neuron 도구](#)를 참조하세요.

Neuron Monitor 사용(neuron-monitor)

Neuron Monitor는 시스템에서 실행되는 Neuron 런타임에서 메트릭을 수집하고 수집된 데이터를 stdout 형식으로 스트리밍합니다. JSON 이러한 지표는구성 파일을 제공하여 구성하는 지표 그룹으로 구성됩니다. Neuron Monitor에 대한 자세한 내용은 [Neuron Monitor사용 설명서](#)를 참조하세요.

Neuron 소프트웨어 업그레이드

내에서 SDK Neuron 소프트웨어를 업데이트하는 방법에 대한 자세한 내용은 다음을 참조하십시오. DLAMI AWS 뉴런 [설정](#) 가이드.

다음 단계

[with 사용 DLAMI AWS Neuron](#)

with 사용 DLAMI AWS Neuron

를 사용하는 일반적인 워크플로 AWS SDKNeuron은 이전에 학습된 기계 학습 모델을 컴파일 서버에서 컴파일하는 것입니다. 그런 다음 아티팩트를 Inf1 인스턴스에 배포하여 실행합니다. AWS Deep Learning AMIs (DLAMI)에는 Inferentia를 사용하는 Inf1 인스턴스에서 추론을 컴파일하고 실행하는 데 필요한 모든 것이 사전 설치되어 있습니다.

다음 섹션에서는 Inferentia와 함께 사용하는 방법을 설명합니다. DLAMI

내용

- [TensorFlow-Neuron 사용 및 AWS 뉴런 컴파일러](#)
- [사용 AWS 뉴런 TensorFlow 서빙](#)
- [MXNet-Neuron을 사용하여 AWS 뉴런 컴파일러](#)
- [MXNet-뉴런 모델 서빙 사용](#)
- [PyTorch-Neuron을 사용하면 AWS 뉴런 컴파일러](#)

TensorFlow-Neuron 사용 및 AWS 뉴런 컴파일러

이 튜토리얼에서는 사용 방법을 보여줍니다. AWS Neuron 컴파일러는 Keras ResNet -50 모델을 컴파일하고 저장된 모델 형식으로 내보내는 데 사용됩니다. SavedModel 이 형식은 모델 호환이 가능한 일반적인 형식입니다. TensorFlow 또한 예제 입력을 사용하여 Inf1 인스턴스에서 추론을 실행하는 방법을 알아봅니다.

뉴런에 대한 자세한 내용은 다음을 참조하십시오. SDK [AWS 뉴런 문서](#). SDK

내용

- [사전 조건](#)
- [Conda 환경 활성화](#)
- [Resnet50 컴파일](#)
- [ResNet5.0 추론](#)

사전 조건

이 자습서를 사용하기 전에 [를 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)의 설정 단계를 완료해야 합니다. 또한 딥 러닝과 사용에 대해서도 잘 알고 있어야 합니다. DLAMI

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 ResNet 50 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
tf.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
```

ResNet5.0 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

[사용 AWS 뉴런 TensorFlow 서빙](#)

사용 AWS 뉴런 TensorFlow 서빙

이 튜토리얼에서는 그래프를 구성하고 그래프를 추가하는 방법을 보여줍니다. AWS Servicing과 함께 TensorFlow 사용할 저장된 모델을 익스포트하기 전의 뉴런 컴파일 단계. TensorFlow 서빙은 네트워크 전반에서 추론을 확장할 수 있는 서빙 시스템입니다. 뉴런 TensorFlow 서빙은 일반 서빙과 동일하게 API 사용합니다. TensorFlow 유일한 차이점은 저장된 모델을 컴파일해야 한다는 것입니다. AWS Inferentia와 진입점은 이름이 다른 바이너리입니다. tensorflow_model_server_neuron 바이너리는 `/usr/local/bin/tensorflow_model_server_neuron` 있으며 사전 설치되어 있습니다. DLAMI

뉴런에 대한 자세한 내용은 SDK 다음을 참조하십시오. [AWS 뉴런 문서](#). [SDK](#)

내용

- [사전 조건](#)
- [Conda 환경 활성화](#)
- [저장된 모델 컴파일 및 내보내기](#)
- [저장된 모델 제공](#)
- [모델 서버에 대한 추론 요청 생성](#)

사전 조건

이 자습서를 사용하기 전에 [틀 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)의 설정 단계를 완료해야 합니다. 또한 딥 러닝과 사용에 대해서도 잘 알고 있어야 합니다. DLAMI

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 DRAM 의해 Inferentia 디바이스에 스테이징됩니다.

```
...
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.py라는 Python 스크립트를 생성합니다. 이 스크립트는 서비스 RPC 프레임워크인 g를 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/aws-labs/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 RPC 명령어와 함께 `g`를 사용하여 모델에서 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-Neuron을 사용하여 AWS 뉴런 컴파일러

MXNet-Neuron 컴파일러는 모델 그래프에서 실행할 수 있는 모델 그래프를 컴파일하는 방법을 API 제공 합니다. AWS 인퍼런시아 디바이스.

이 예제에서는 `tf`를 사용하여 ResNet -50 모델을 컴파일하고 이를 사용하여 API 추론을 실행합니다.

SDKNeuron에 대한 자세한 내용은 다음을 참조하십시오. [AWS 뉴런 문서](#). [SDK](#)

내용

- [사전 조건](#)
- [Conda 환경 활성화](#)
- [Resnet50 컴파일](#)
- [ResNet5.0 추론](#)

사전 조건

이 자습서를 사용하기 전에 [를 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)의 설정 단계를 완료해야 합니다. 또한 딥 러닝과 사용에 대해서도 잘 알고 있어야 합니다. DLAMI

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 API Python을 사용하여 -50 모델을 컴파일합니다 ResNet.

```
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
```

ResNet5.0 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/awsmlabs/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-뉴런 모델 서빙 사용](#)

MXNet-뉴런 모델 서빙 사용

이 자습서에서는 사전 훈련된 MXNet 모델을 사용하여 다중 모델 서버 () MMS 로 실시간 이미지 분류를 수행하는 방법을 배웁니다. MMS기계 학습 또는 딥 러닝 프레임워크를 사용하여 학습된 딥 러닝 모델을 제공하기 위한 유연한 easy-to-use 도구입니다. 이 자습서에는 다음을 사용하는 컴파일 단계가 포함되어 있습니다. AWS 뉴런 및 사용 MMS 구현 MXNet

뉴런에 대한 자세한 내용은 SDK 다음을 참조하십시오. [AWS 뉴런 문서. SDK](#)

내용

- [사전 조건](#)
- [Conda 환경 활성화](#)
- [예제 코드 다운로드](#)
- [모델 컴파일](#)
- [추론 실행](#)

사전 조건

이 자습서를 사용하기 전에 [블 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)의 설정 단계를 완료해야 합니다. 또한 딥 러닝과 사용에 대해서도 잘 알고 있어야 합니다. DLAMI

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/aws-labs/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 ResNet 50 모델을 Inferentia 장치 타겟으로 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 예측 클래스의 ImageNet 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 `and Gluon`을 지원하지 `model_input` 않기 때문에 불필요한 데이터 사본을 주석 처리해야 합니다 `NDArray`. APIs

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

다음 명령을 사용하여 `model-archiver`로 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 RESTful API 사용하여 를 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 `rm` 통해 삭제 명령을 실행하고 다음 명령을 사용하여 모델 서버를 중지하십시오.
RESTful API

```

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-Neuron을 사용하면 AWS 뉴런 컴파일러

PyTorch-Neuron 컴파일은 모델 그래프에서 실행할 수 있는 모델 그래프를 컴파일하는 방법을 API 제 공합니다. AWS 인퍼런시아 디바이스.

훈련된 모델은 Inf1 인스턴스에서 배포하기 전에 Inferentia 대상에 컴파일되어야 합니다. 다음 튜토리 얼은 토치비전 ResNet 50 모델을 컴파일하고 저장된 모듈로 내보냅니다. TorchScript 이러한 모델은 추론을 실행하는 데 사용됩니다.

편의상 이 자습서에서는 컴파일 및 추론 모두에 Inf1 인스턴스를 사용합니다. 실제로는 c5 인스턴스 패 밀리와 같은 다른 인스턴스 유형을 사용하여 모델을 컴파일할 수 있습니다. 그런 다음 컴파일된 모델 을 Inf1 추론 서버에 배포해야 합니다. 자세한 내용은 [단원을 참조하십시오.AWS 뉴런 문서. PyTorch SDK](#)

내용

- [사전 조건](#)
- [Conda 환경 활성화](#)
- [Resnet50 컴파일](#)

- [ResNet5.0 추론](#)

사전 조건

이 자습서를 사용하기 전에 [블 사용하여 DLAMI 인스턴스 시작 AWS Neuron](#)의 설정 단계를 완료해야 합니다. 또한 딥 러닝과 사용에 대해서도 잘 알고 있어야 합니다. DLAMI

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 API Python을 사용하여 -50 모델을 컴파일합니다 ResNet.

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.

ResNet5.0 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']
```

더 ARM64 DLAMI

AWS ARM64GPU DLAMIs 딥 러닝 워크로드에 고성능 및 비용 효율성을 제공하도록 설계되었습니다. 특히, G5G 인스턴스 유형에는 ARM64 기반이 포함됩니다. [AWS Graviton2 프로세서는 처음부터 다음과 같이 개발되었습니다.](#) AWS 또한 고객이 클라우드에서 워크로드를 실행하는 방식에 최적화되었습니다. AWS ARM64GPU DLAMIs Docker, Docker, NVIDIA NVIDIA Driver, Cu CUDA DNN NCCL, 및 와 같은 인기 있는 기계 학습 프레임워크로 사전 구성되어 있습니다. TensorFlow PyTorch

G5G 인스턴스 유형을 사용하면 Graviton2의 가격 및 성능 이점을 활용하여 가속화된 x86 기반 인스턴스와 비교할 때 훨씬 저렴한 비용으로 GPU 가속화된 딥 러닝 모델을 배포할 수 있습니다. GPU

다음 중 하나를 선택하세요. ARM64 DLAMI

원하는 인스턴스로 [G5G 인스턴스를](#) 시작합니다. ARM64 DLAMI

실행에 대한 step-by-step 지침은 [시작 및 구성을 참조하십시오. DLAMI DLAMI](#)

최신 ARM64 DLAMIs 버전 목록은 의 [릴리스 노트를 참조하십시오 DLAMI.](#)

시작하기

다음 항목에서는 를 사용하여 시작하는 방법을 보여줍니다 ARM64DLAMI.

내용

- [사용 ARM64 GPU PyTorch DLAMI](#)

사용 ARM64 GPU PyTorch DLAMI

더 AWS Deep Learning AMIs Arm64 프로세서 기반과 함께 바로 사용할 수 GPU가 있으며 최적화되어 제공됩니다. PyTorch ARM64GPU PyTorch DLAMI 여기에는 [PyTorch TorchVision](#), 로 사전 구성된 Python 환경과 딥 러닝 교육 및 추론 사용 [TorchServe](#) 사례에 사용할 수 있는 Python 환경이 포함됩니다.

내용

- [PyTorch Python 환경 확인하기](#)
- [다음을 사용하여 교육 샘플을 실행하십시오. PyTorch](#)

- [를 사용하여 추론 샘플을 실행합니다. PyTorch](#)

PyTorch Python 환경 확인하기

G5g 인스턴스에 연결하고 다음 명령을 사용하여 기본 Conda 환경을 활성화합니다.

```
source activate base
```

명령 프롬프트에 PyTorch TorchVision, 및 기타 라이브러리가 포함된 기본 Conda 환경에서 작업 중임을 나타내야 합니다.

```
(base) $
```

PyTorch 환경의 기본 도구 경로를 확인하십시오.

```
(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)
```

다음을 사용하여 교육 샘플을 실행하십시오. PyTorch

샘플 MNIST 교육 작업 실행:

```
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%)
```

를 사용하여 추론 샘플을 실행합니다. PyTorch

다음 명령을 사용하여 사전 학습된 densenet161 모델을 다운로드하고 다음을 사용하여 추론을 실행합니다. TorchServe

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

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

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

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

# Wait for the model server to start
sleep 30

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

출력은 다음과 비슷한 형태가 됩니다.

```
{
  "tiger_cat": 0.4693363308906555,
```



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

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 섹션에서는 DLAMI의 프레임워크와 도구를 사용하여 추론을 실행하는 방법에 대한 자습서를 제공합니다.

추론 도구

- [TensorFlow 서빙](#)

모델 제공

다음은 Conda를 AMI 사용한 딥 러닝에 설치된 모델 제공 옵션입니다. 옵션 중 한 가지를 클릭하여 사용법을 알아보십시오.

주제

- [TensorFlow 서빙](#)
- [TorchServe](#)

TensorFlow 서빙

[TensorFlow 서빙](#)은 머신 러닝 모델을 위한 유연한 고성능 서비스 시스템입니다.

Conda를 사용한 딥 러닝과 함께 사전 tensorflow-serving-api 설치되어 AMI 있습니다! MNIST 모델을 학습시키고, 내보내고, 제공할 수 있는 예제 스크립트를 찾을 수 있습니다. ~/examples/tensorflow-serving/

이러한 예제를 실행하려면 먼저 Conda를 사용하여 딥 AMI 러닝에 연결하고 TensorFlow 환경을 활성화하십시오.

```
$ source activate tensorflow2_p310
```

이제 서비스할 예제 스크립트가 있는 폴더로 이동합니다.

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

Pretrained Inception Model 제공

다음은 Inception과 같은 다른 모델을 제공하기 위해 시도할 수 있는 예제입니다. 일반적으로 서빙 가능한 모델과 클라이언트 스크립트가 이미 다운로드되어 있어야 합니다. DLAMI

Inception 모델을 사용하여 추론 제공 및 테스트

1. 이 모델을 다운로드합니다.

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

2. 이 모델의 압축을 해제합니다.

```
$ unzip INCEPTION.zip
```

3. 허스키 그림을 다운로드합니다.

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

4. 서버를 시작합니다. Amazon Linux의 경우 model_base_path에 사용되는 디렉터리를 /home/ubuntu에서 /home/ec2-user로 변경해야 합니다.

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

5. 서버가 포그라운드에서 실행 중일 때 계속하려면 다른 터미널 세션을 시작해야 합니다. 새 터미널을 열고 tensorflow 사용하여 source activate tensorflow2_p310 활성화하십시오. 그

런 다음 원하는 텍스트 편집기를 사용하여 다음 내용이 들어 있는 스크립트를 생성합니다. 이름을 `inception_client.py`로 지정합니다. 이 스크립트는 이미지 파일 이름을 파라미터로 받아서 사전 교육 모델에서 예측 결과를 가져옵니다.

```
from __future__ import print_function

import grpc
import tensorflow as tf
import argparse

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

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

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

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

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

```
print("Inception Client Passed")

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

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

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

MNIST모델 교육 및 제공

이 자습서에서는 모델을 하나 내보낸 다음 `tensorflow_model_server` 애플리케이션으로 서비스해 보겠습니다. 마지막으로, 예제 클라이언트 스크립트로 모델 서버를 테스트할 수 있습니다.

MNIST모델을 학습시키고 내보낼 스크립트를 실행합니다. 이 스크립트의 유일한 인수인 폴더 위치를 입력해야 모델이 저장됩니다. 여기서는 그냥 `mnist_model`이라고 입력하겠습니다. 그러면 스크립트가 폴더를 만들어 줍니다.

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

이 스크립트에서 결과가 출력되려면 시간이 걸리므로 인내심을 가지고 기다리십시오. 학습이 완료되어 모델을 최종적으로 내보내면 다음과 같은 결과를 보게 됩니다.

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

다음 단계는 `tensorflow_model_server`를 실행하여 내보낸 모델에 서비스를 제공하는 것입니다.

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

서버를 테스트하기 위한 클라이언트 스크립트가 준비되어 있습니다.

테스트해 보려면 새 터미널 창을 열어야 합니다.

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

더 많은 기능과 예제

Servicing에 대해 더 자세히 TensorFlow 알아보려면 [TensorFlow 웹 사이트](#)를 확인하세요.

[Amazon Elastic TensorFlow Inference](#)와 함께 서비스를 사용할 수도 있습니다. 자세한 내용은 [Servicing과 함께 TensorFlow Elastic Inference를 사용하는](#) 방법에 대한 가이드를 확인하세요.

TorchServe

TorchServe 에서 PyTorch 익스포트한 딥 러닝 모델을 제공하기 위한 유연한 도구입니다. TorchServe Conda를 통한 딥 러닝과 AMI 함께 사전 설치되어 제공됩니다.

사용에 TorchServe 대한 자세한 내용은 [설명서용 PyTorch 모델 서버](#)를 참조하십시오.

주제

이미지 분류 모델 제공: TorchServe

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

이미지 분류 모델의 예를 제공하려면 다음을 참조하십시오. TorchServe

1. Conda v34 이상을 사용하여 딥 러닝을 AMI 사용하여 Amazon Elastic Compute Cloud (AmazonEC2) 인스턴스에 연결하십시오.
2. pytorch_p310 환경을 활성화합니다.

```
source activate pytorch_p310
```

3. TorchServe 리포지토리를 복제한 다음 모델을 저장할 디렉토리를 생성하십시오.

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

4. 모델 아카이버를 사용하여 모델을 보관합니다. extra-files매개변수는 TorchServe 리포지토리의 파일을 사용하므로 필요한 경우 경로를 업데이트하세요. 모델 아카이버에 대한 자세한 내용은 [Torch Model 아카이버](#)를 참조하십시오. TorchServe

```
wget https://download.pytorch.org/models/densenet161-8d451a50.pth
```

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

5. `torch-model-archiver` 실행하여 엔드포인트를 시작합니다. `> /dev/null`을 추가하면 로그 출력이 중지됩니다.

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

6. 새끼 고양이 이미지를 다운로드하여 TorchServe 예측 엔드포인트로 전송하십시오.

```
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 5개 JSON 예측과 비슷한 예측을 반환합니다. 이미지에 이집트 고양이 있을 확률은 47% 이고 얼룩무늬 고양이가 있을 확률은 46% 입니다.

```
{
  "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

TorchServe 인스턴스에서 실행할 수 있는 다양한 예제가 있습니다. DLAMI [TorchServe 프로젝트 리포지토리 예제 페이지](#)에서 확인할 수 있습니다.

추가 정보

Docker를 TorchServe 사용하여 설정하는 방법 및 최신 TorchServe 기능을 포함한 자세한 TorchServe 설명서는 [의 TorchServe 프로젝트 페이지](#)를 참조하십시오. GitHub

DLAMI 업그레이드

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

주제

- [최신 버전으로 DLAMI 업그레이드](#)
- [소프트웨어 업데이트에 관한 팁](#)
- [새 업데이트에 대한 알림 받기](#)

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 생성](#)을 참조하세요.
2. 새로 생성한 Amazon EBS 볼륨을 기존 DLAMI에 연결합니다. 자세한 지침은 [Amazon EBS 볼륨 연결](#)을 참조하세요.
3. 데이터(예: 데이터 세트, 체크포인트, 구성 파일)를 이전합니다.
4. DLAMI를 실행합니다. 자세한 지침은 [DLAMI 시작 및 구성](#) 섹션을 참조하세요.
5. 기존 DLAMI에서 아마존 EBS 볼륨을 분리합니다. 자세한 지침은 [Amazon EBS 볼륨 분리](#)를 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 [정리](#) 섹션을 참조하세요.

소프트웨어 업데이트에 관한 팁

때때로 DLAMI에서 소프트웨어를 수동으로 업데이트해야 하는 경우가 있을 수 있습니다. 일반적으로 Python 패키지 업데이트에는 pip을 사용하는 것이 좋습니다. 또한 Conda를 사용하는 DLAMI의 Conda 환경에서 패키지를 업데이트할 때는 pip을 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 오류가 발생할 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 AMIs에는 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 업데이트에 대한 알림 받기

Note

AWS 딥러닝 AMI는 보안 패치에 대해 매주 릴리스 주기를 적용합니다. 공식 릴리스 노트에는 포함되지 않을 수 있지만 이러한 증분 보안 패치에 대해서는 릴리스 알림이 전송됩니다.

새 DLAMI가 릴리스될 때마다 알림을 받을 수 있습니다. 다음 주제에 대한 알림이 [Amazon SNS](#)에 게시됩니다.


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

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

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

1. [Amazon SNS 콘솔](#)을 엽니다.
2. 필요한 경우 탐색 표시줄에서 AWS 지역을 미국 서부 (오레곤) 로 변경합니다. 구독 중인 SNS 알림이 생성된 지역을 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 복사하여 붙여넣습니다.
arn:aws:sns:us-west-2:767397762724:dlami-updates
 - b. 프로토콜의 경우 [Amazon SQS, AWS Lambda, 이메일, 이메일-JSON] 중에서 하나를 선택합니다.
 - c. Endpoint에 알림을 수신하는 데 사용할 리소스의 이메일 주소 또는 Amazon 리소스 이름 (ARN) 을 입력합니다.
 - d. 구독 생성을 선택합니다.
5. AWS 알림 - 구독 확인이라는 제목이 포함된 확인 이메일을 받게 됩니다. 이메일을 열고 구독 확인을 선택하여 구독 신청을 완료합니다.

보안 속의 AWS Deep Learning AMIs

클라우드 보안: AWS 최우선 과제입니다.로서 AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 두 회사 간의 공동 책임입니다. AWS 그리고 당신. [공동 책임 모델](#)은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 — AWS 실행 중인 인프라를 보호할 책임이 있습니다. AWS의 서비스 AWS 클라우드. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 제3자 감사자는 보안 조치의 일환으로 당사 보안의 효과를 정기적으로 테스트하고 확인합니다. [AWS 규정 준수 프로그램](#). 적용되는 규정 준수 프로그램에 대한 자세한 내용은 DLAMI 다음을 참조하십시오. [AWS 규정 준수 프로그램별 범위 내 서비스](#).
- 클라우드에서의 보안 — 귀하의 책임은 다음에 의해 결정됩니다. AWS 사용하는 서비스. 또한 귀하는 귀사의 데이터의 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 공동 책임 모델을 사용할 때 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 항목에서는 보안 및 규정 준수 목표를 DLAMI 충족하도록 구성하는 방법을 보여줍니다. 또한 다른 방법을 사용하는 방법도 배웁니다. AWS DLAMI 리소스를 모니터링하고 보호하는 데 도움이 되는 서비스.

자세한 내용은 [Amazon의 보안을 참조하십시오](#) EC2.

주제

- [내 데이터 보호 AWS Deep Learning AMIs](#)
- [Identity 및 Access Management의 AWS Deep Learning AMIs](#)
- [로그인 및 모니터링 AWS Deep Learning AMIs](#)
- [규정 준수 검증: AWS Deep Learning AMIs](#)
- [레질리언스: AWS Deep Learning AMIs](#)
- [의 인프라 보안 AWS Deep Learning AMIs](#)

내 데이터 보호 AWS Deep Learning AMIs

The AWS [공동 책임 모델](#): 다음과 같은 데이터 보호에 적용됩니다. AWS 딥 러닝AMI. 이 모델에 설명된 바와 같이 AWS 모든 시스템을 운영하는 글로벌 인프라를 보호하는 책임이 있습니다. AWS 클라우

드. 이 인프라에서 호스팅되는 콘텐츠에 대한 통제권을 유지할 책임은 귀하에게 있습니다. 또한 귀하에 대한 보안 구성 및 관리 작업을 담당합니다. AWS 서비스 사용하는 것. 데이터 프라이버시에 대한 자세한 내용은 [데이터 프라이버시를 참조하십시오](#) FAQ. 유럽의 데이터 보호에 대한 자세한 내용은 다음을 참조하십시오. [AWS 공동 책임 모델 및 관련 GDPR](#) 블로그 게시물 AWS 보안 블로그.

데이터 보호를 위해 다음을 보호하는 것이 좋습니다. AWS 계정 자격 증명 및 개별 사용자 설정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM). 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정마다 다단계 인증 (MFA) 을 사용하십시오.
- SSL/를 사용하여 다음과 TLS 통신할 수 있습니다. AWS 있습니다. TLS1.2가 필요하고 TLS 1.3을 권장합니다.
- 다음을 사용하여 사용자 활동 API 로깅을 설정하고 사용자 활동을 기록합니다. AWS CloudTrail. CloudTrail 트레일을 사용하여 캡처하는 방법에 대한 자세한 내용은 AWS 활동에 대한 자세한 내용은 [CloudTrail 트레일 사용](#) 참조하십시오. AWS CloudTrail 사용자 가이드.
- 사용 AWS 암호화 솔루션 및 포함된 모든 기본 보안 제어 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 액세스 시 FIPS 140-3개의 검증된 암호화 모듈이 필요한 경우 AWS 명령줄 인터페이스 또는 API an 을 통해 엔드포인트를 사용하십시오. FIPS 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [연방 정보 처리 표준 \(FIPS\) 140-3](#)을 참조하십시오.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 다른 회사와 DLAMI 협력하는 경우가 포함됩니다. AWS 서비스 콘솔 사용API, AWS CLI, 또는 AWS SDKs. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 제공하는 경우 해당 서버에 대한 요청을 URL 검증하기 위해 자격 증명 정보를 에 포함하지 않는 것이 좋습니다. URL

Identity 및 Access Management의 AWS Deep Learning AMIs

AWS Identity and Access Management (IAM) 는 AWS 서비스 이를 통해 관리자는 다음 항목에 대한 액세스를 안전하게 제어할 수 있습니다. AWS 있습니다. IAM관리자는 DLAMI 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. IAM는 AWS 서비스 추가 비용 없이 사용할 수 있습니다.

ID 및 액세스 관리에 대한 자세한 내용은 [Amazon의 ID 및 액세스 관리를 참조하십시오](#)EC2.

주제

- [자격 증명을 통한 인증](#)
- [정책을 사용하여 액세스 관리](#)
- [IAM아마존과 함께 EMR](#)

자격 증명을 통한 인증

인증은 로그인하는 방법입니다. AWS ID 자격 증명 사용 인증 (로그인) 을 받아야 합니다. AWS다음과 같이) AWS 계정 루트 사용자 IAM사용자로서, 또는 IAM 역할을 맡아서

에 로그인할 수 있습니다. AWS ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 사용할 수 있습니다. AWS IAM Identity Center 페더레이션 ID의 예로는 (IAMID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명입니다. 페더레이션 ID로 로그인하는 경우 관리자는 이전에 역할을 사용하여 ID 페더레이션을 설정했습니다. IAM 액세스하는 경우 AWS 페더레이션을 사용하면 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 로그인할 수 있습니다. AWS Management Console 또는 AWS 액세스 포털. 로그인에 대한 자세한 내용은 AWS로그인하는 [방법을 참조하십시오](#). AWS 계정의 AWS 로그인 사용자 가이드.

액세스하는 경우 AWS 프로그래밍 방식으로 AWS 자격 증명을 사용하여 요청에 암호로 서명할 수 있는 소프트웨어 개발 키트 (SDKCLI) 와 명령줄 인터페이스 () 를 제공합니다. 사용하지 않는 경우 AWS 도구를 사용하려면 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 [서명을 참조하십시오](#). AWS APIIAM사용 설명서의 요청.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예: AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 의 [다단계 인증을 참조하십시오](#). AWS IAM Identity Center 사용 설명서 및 다단계 인증 [사용 \(\) MFA AWS](#)(출처: IAM 사용 설명서).

AWS 계정 루트 사용자

를 생성할 때 AWS 계정모든 계정에 완전히 액세스할 수 있는 하나의 로그인 ID로 시작합니다. AWS 서비스 및 계정 내 리소스 이 ID를 다음과 같이 부릅니다. AWS 계정 루트 사용자는 계정을 만들 때 사용한 이메일 주소와 암호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는

작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 작업의 전체 목록은 [사용 설명서의 루트 사용자 자격 증명이 필요한 작업을](#) 참조하십시오. IAM

IAM 사용자 및 그룹

[IAM 사용자](#)는 내 ID 내에 있는 ID입니다. AWS 계정 이는 한 사람이나 애플리케이션에 대한 특정 권한을 가지고 있습니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명을 가진 IAM 사용자를 만드는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 특정 사용 사례에서 IAM 사용자의 장기 자격 증명이 필요한 경우에는 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 [사용 설명서의 장기 자격 증명이 필요한 사용 사례에 대한 정기적인 액세스 키 IAM 교체](#)를 참조하십시오.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 ID입니다. 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 이름을 지정한 IAMAdmins 그룹을 만들고 해당 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세히 알아보려면 [사용 설명서의 역할 대신 IAM 사용자를 만드는 시기](#)를 참조하십시오. IAM

IAM 역할

[IAM 역할](#)은 내 안의 정체성입니다. AWS 계정 여기에는 특정 권한이 있습니다. 사용자와 비슷하지만 특정 IAM 사용자와는 관련이 없습니다. 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS Management Console [역할을 바꿔서 맡이](#)조. 를 호출하여 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 오퍼레이션을 사용하거나 사용자 지정을 사용합니다 URL. 역할 사용 방법에 대한 자세한 내용은 [사용 IAM 설명서의 IAM 역할 사용](#)을 참조하십시오.

IAM 임시 자격 증명이 있는 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. 페더레이션을 위한 역할에 대한 자세한 내용은 [IAM 사용 설명서의 타사 ID 제공자를 위한 역할 생성](#)을 참조하십시오. IAM Identity Center를 사용하는 경우 권한 집합을 구성합니다. ID가 인증된 후 액세스할 수 있는 대상을 제어하기 위해 IAM Identity Center는 권한 집합을 역할의 상관 관계와 연결합니다. IAM 권한 집합에 대한 자세한 내용은 [권한 집합의 사용](#) [권한](#) 집합을 참조하십시오. AWS IAM Identity Center 사용 설명서.
- 임시 IAM 사용자 권한 — IAM 사용자 또는 역할은 역할을 맡아 특정 작업에 대해 일시적으로 다른 권한을 부여받을 수 있습니다. IAM

- 계정 간 액세스 - IAM 역할을 사용하여 다른 계정의 사용자 (신뢰할 수 있는 사용자) 가 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 하지만 일부 경우에는 AWS 서비스 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 계정 간 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 사용 설명서의 [IAM 계정 간 리소스 액세스](#)를 참조하십시오. IAM
- 서비스 간 액세스 — 일부 AWS 서비스 다른 기능 사용 AWS 서비스. 예를 들어, 서비스를 호출하면 해당 서비스가 Amazon에서 애플리케이션을 EC2 실행하거나 Amazon S3에 객체를 저장하는 것이 일반적입니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
- 전달 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 AWS, 귀하는 주도자로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS전화를 건 주체의 권한을 사용합니다. AWS 서비스, 요청과 결합 AWS 서비스 다운스트림 서비스에 요청할 수 있습니다. FAS요청은 서비스가 다른 서비스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. AWS 서비스 또는 완료해야 할 리소스. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS요청 시 적용되는 정책 세부 정보는 [전달 액세스 세션](#)을 참조하십시오.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 간주하는 [IAM 역할입니다](#). IAM관리자는 내부에서 IAM 서비스 역할을 만들고, 수정하고, 삭제할 수 있습니다. 자세한 내용은 권한을 위임하기 위한 역할 [만들기를 참조하십시오. AWS 서비스](#)(출처: IAM 사용 설명서).
- 서비스 연결 역할 - 서비스 연결 역할은 다음과 연결된 서비스 역할 유형입니다. AWS 서비스. 서비스가 사용자를 대신하여 작업을 수행하는 역할을 맡을 수 있습니다. 서비스 연결 역할은 다음과 같습니다. AWS 계정 서비스가 소유합니다. IAM관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.
- Amazon에서 실행되는 애플리케이션 EC2 — IAM 역할을 사용하여 EC2 인스턴스에서 실행 중이고 다음을 생성하는 애플리케이션에 대한 임시 자격 증명을 관리할 수 있습니다. AWS CLI 또는 AWS API요청. EC2인스턴스 내에 액세스 키를 저장하는 것보다 이 방법이 더 좋습니다. 할당하려면 AWS EC2인스턴스에 역할을 부여하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며, 이를 통해 EC2 인스턴스에서 실행 중인 프로그램이 임시 자격 증명을 얻을 수 있습니다. 자세한 내용은 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 IAM](#) 참조하십시오.

IAM 역할을 사용할지 IAM 사용자를 사용할지 알아보려면 사용 [설명서의 IAM 역할 생성 시기\(사용자 대신\)](#) 를 IAM참조하십시오.

정책을 사용하여 액세스 관리

에서 액세스를 제어할 수 있습니다. AWS 정책을 생성하여 정책에 연결하면 됩니다. AWS ID 또는 리소스 정책은 다음의 객체입니다. AWS 이는 ID 또는 리소스와 연결될 경우 해당 권한을 정의합니다. AWS 보안 주체 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 다음 위치에 저장됩니다. AWS JSON문서로. JSON정책 문서의 구조 및 내용에 대한 자세한 내용은 IAM사용 [설명서의 JSON 정책 개요](#)를 참조하십시오.

관리자는 다음을 사용할 수 있습니다. AWS JSON정책을 통해 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. IAM관리자는 IAM 정책을 생성하여 필요한 리소스에서 작업을 수행할 수 있는 권한을 사용자에게 부여할 수 있습니다. 그러면 관리자가 역할에 IAM 정책을 추가할 수 있으며, 사용자는 역할을 수입할 수 있습니다.

IAM정책은 작업을 수행하는 데 사용하는 방법에 관계없이 작업에 대한 권한을 정의합니다. 예를 들어, iam:GetRole 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 에서 역할 정보를 가져올 수 있습니다. AWS Management Console, AWS CLI, 또는 AWS API.

자격 증명 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. ID 기반 정책을 만드는 방법을 알아보려면 사용 설명서의 [IAM정책 생성](#)을 참조하십시오. IAM

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 조직의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정. 관리형 정책에는 다음이 포함됩니다. AWS 관리형 정책 및 고객 관리형 정책. 관리형 정책과 인라인 정책 중에서 선택하는 방법을 알아보려면 IAM사용 설명서의 [관리형 정책과 인라인 정책 중 선택](#)을 참조하십시오.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예로는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스

의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 또는 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 사용할 수 없습니다. AWS 리소스 기반 정책의 관리형 정책 IAM

액세스 제어 목록 (ACLs)

액세스 제어 목록 (ACLs)은 리소스에 액세스할 수 있는 권한을 가진 주체 (계정 구성원, 사용자 또는 역할)를 제어합니다. ACLs정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 JSON 비슷합니다.

아마존 S3, AWS WAF, VPC Amazon은 지원하는 서비스의 예입니다ACLs. 자세한 내용은 Amazon 심플 스토리지 서비스 개발자 안내서의 [액세스 제어 목록 \(ACL\) 개요](#)를 참조하십시오. ACLs

기타 정책 유형

AWS 일반적이지 않은 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 정책이 IAM 엔티티 (IAM사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 자격 증명 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 내용은 사용 IAM설명서의 [IAM 엔티티에 대한 권한 경계를](#) 참조하십시오.
- 서비스 제어 정책 (SCPs) — SCPs 조직 또는 OU (조직 구성 단위)에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations. AWS Organizations 여러 개를 그룹화하고 중앙에서 관리하는 서비스입니다. AWS 계정 귀사가 소유한 것입니다. 조직의 모든 기능을 사용하도록 설정하면 일부 또는 모든 계정에 서비스 제어 정책 (SCPs)을 적용할 수 있습니다. 각 항목을 포함하여 구성원 계정의 엔티티에 대한 권한을 SCP 제한합니다. AWS 계정 루트 사용자. Organizations 및 SCPs에 대한 자세한 내용은 의 [서비스 제어 정책을](#) 참조하십시오. AWS Organizations 사용 설명서.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM사용 설명서의 [세션 정책을](#) 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 방법을 알아보려면 AWS 여러 정책 유형이 관련된 경우 요청을 허용할지 여부를 결정하려면 IAM사용 설명서의 [정책 평가 로직](#)을 참조하십시오.

IAM아마존과 함께 EMR

다음을 사용할 수 있음: AWS Identity and Access Management EMRAmazon을 사용하여 사용자를 정의하면 AWS 리소스, 그룹, 역할, 정책 또한 어느 것을 제어할 수 있는지 제어할 수 있습니다. AWS 이러한 사용자 및 역할이 액세스할 수 있는 서비스.

Amazon에서 사용하는 IAM 방법에 대한 자세한 내용은 EMR 을 참조하십시오. [AWS Amazon용 ID 및 액세스 관리 EMR](#).

로그인 및 모니터링 AWS Deep Learning AMIs

귀하의 AWS Deep Learning AMIs 인스턴스는 Amazon에 GPU 사용 통계를 보고하는 유틸리티를 비롯한 여러 GPU 모니터링 도구와 함께 제공됩니다 CloudWatch. 자세한 내용은 [GPU모니터링 및 최적화](#)와 [Amazon 모니터링](#)을 참조하십시오EC2.

사용량 추적

다음 내용은 다음과 같습니다. AWS Deep Learning AMIs 운영 체제 배포판에는 다음을 허용하는 코드가 포함됩니다. AWS 인스턴스 유형, 인스턴스 ID, DLAMI 유형 및 OS 정보를 수집합니다. 내에서 사용되는 명령에 대한 DLAMI 정보는 수집되거나 보관되지 않습니다. 에 대한 다른 DLAMI 정보는 수집되거나 보관되지 않습니다.

- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Amazon Linux 2

에 대한 사용 추적을 거부하려면 시작 시 Amazon EC2 인스턴스에 태그를 추가하십시오. DLAMI 태그는 관련 값이 OPT_OUT_TRACKING으로 설정된 true 키를 사용해야 합니다. 자세한 내용은 [Amazon EC2 리소스 태그 지정](#)을 참조하십시오.

규정 준수 검증: AWS Deep Learning AMIs

제3자 감사자는 다음 기관의 보안 및 규정 준수를 평가합니다. AWS Deep Learning AMIs 여러 항목의 일부로 AWS 규정 준수 프로그램. 지원되는 규정 준수 프로그램에 대한 자세한 내용은 [Amazon의 규정 준수 검증을](#) 참조하십시오EC2.

목록은 다음과 같습니다. AWS 특정 규정 준수 프로그램 범위의 서비스는 다음을 참조하십시오. [AWS 규정 준수 프로그램별 범위 내 서비스](#). 일반 정보는 다음을 참조하십시오. [AWS 규정 준수 프로그램](#).

다음은 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. AWS Artifact. 자세한 내용은 보고서 [다운로드를 참조하십시오. AWS](#).

사용 시 규정 준수 DLAMI 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- [보안 및 규정 준수 킷스타트 가이드](#) - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다. AWS.
- [AWS 규정 준수 리소스](#) — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- 다음 [규칙을 사용하여 리소스를 평가합니다](#). AWS Config 개발자 가이드 — The AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- [AWS Security Hub](#)— 이것은 AWS 이 서비스는 내부 보안 상태를 포괄적으로 보여줍니다. AWS 이를 통해 보안 업계 표준 및 모범 사례 준수 여부를 확인할 수 있습니다.

레질리언스: AWS Deep Learning AMIs

The AWS 글로벌 인프라는 다음을 중심으로 구축됩니다. AWS 지역 및 가용 영역. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 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.

에 대한 자세한 내용은 AWS 지역 및 가용 영역을 참조하십시오. [AWS 글로벌 인프라](#).

데이터 복원력 및 백업 요구 사항을 지원하는 데 도움이 되는 기능에 대한 자세한 내용은 [Amazon의 복원력을](#) 참조하십시오. EC2

의 인프라 보안 AWS Deep Learning AMIs

의 인프라 보안 AWS Deep Learning AMIs Amazon의 지원을 받습니다EC2. 자세한 내용은 [Amazon의 인프라 보안을 참조하십시오](#)EC2.

DLAMI프레임워크 지원 정책

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

다음과 같은 DLAMI 프레임워크 목록을 보려면 AWS 현재 지원합니다. [DLAMIFramework 지원 정책](#) 페이지를 참조하십시오. 해당 페이지의 표에서 다음 사항을 염두에 두십시오.

- 현재 버전은 프레임워크 버전을 x.y.z 형식으로 지정합니다. 이 형식에서 x는 메이저 버전, y는 마이너 버전, z는 패치 버전을 나타냅니다. 예를 들어 TensorFlow 2.10.1의 경우 메이저 버전은 2, 마이너 버전은 10, 패치 버전은 1입니다.
- 패치 종료 후에는 기간이 지정됩니다. AWS 해당 프레임워크 버전을 지원합니다.

특정 항목에 대한 자세한 내용은 DLAMIs 을 참조하십시오 [DLAMIs 릴리스 정보](#).

DLAMI프레임워크 지원 FAQs

- [어떤 프레임워크 버전에 보안 패치가 적용되나요?](#)
- [이미지는 어떤 역할을 하나요? AWS 새 프레임워크 버전이 출시되면 게시되나요?](#)
- [새로 추가되는 이미지/ SageMaker AWS 특징?](#)
- [지원되는 프레임워크 표에서 현재 버전을 어떻게 확인하나요?](#)
- [지원되는 프레임워크 표에 없는 버전을 실행 중인 경우에는 어떻게 해야 하나요?](#)
- [의 이전 버전을 DLAMIs 지원하나요 TensorFlow?](#)
- [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 계속 up-to-date 확인하려면 어떻게 해야 합니까?](#)
- [Anaconda 리포지토리를 사용하려면 상용 라이선스가 필요한가요?](#)

어떤 프레임워크 버전에 보안 패치가 적용되나요?

프레임워크 버전에 Support라는 레이블이 붙어 있는 경우 [AWS Deep Learning AMIs Framework Support Policy 테이블](#), 보안 패치를 가져옵니다.

이미지는 어떤 역할을 하나요? AWS 새 프레임워크 버전이 출시되면 게시되나요?

TensorFlow 및 의 새 버전이 DLAMIs PyTorch 출시되면 곧 새 버전을 게시합니다. 여기에는 메이저 버전, 메이저 마이너 버전, 프레임워크 버전이 포함됩니다 major-minor-patch . 또한 새 버전의 드라이버와 라이브러리가 출시되면 이미지도 업데이트됩니다. 이미지 유지 관리에 대한 자세한 내용은 [내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?](#) 섹션을 참조하세요.

새로 추가되는 이미지/ SageMaker AWS 특징?

새 기능은 일반적으로 DLAMIs for PyTorch 및 의 최신 버전으로 TensorFlow 출시됩니다. 새 이미지 SageMaker 또는 이미지에 대한 자세한 내용은 특정 이미지의 릴리스 노트를 참조하십시오. AWS 특징. 사용 가능한 DLAMIs 목록은 [릴리스 노트를 참조하십시오 DLAMI](#). 이미지 유지 관리에 대한 자세한 내용은 [내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?](#) 섹션을 참조하세요.

지원되는 프레임워크 표에서 현재 버전을 어떻게 확인하나요?

최신 버전은 [AWS Deep Learning AMIs Framework Support Policy 표](#)는 다음과 같은 최신 프레임워크 버전을 나타냅니다. AWS 에서 사용할 수 GitHub 있습니다. 각 최신 릴리스에는 의 드라이버, 라이브러리 및 관련 패키지에 대한 업데이트가 포함됩니다 DLAMI. 이미지 유지 관리에 대한 자세한 내용은 [내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?](#) 섹션을 참조하세요.

지원되는 프레임워크 표에 없는 버전을 실행 중인 경우에는 어떻게 해야 하나요?

에 없는 버전을 실행 중인 경우 [AWS Deep Learning AMIs Framework Support Policy 표](#)에는 가장 업데이트된 드라이버, 라이브러리 및 관련 패키지가 없을 수 있습니다. 더 많은 up-to-date 버전을 원하면 선택한 최신 DLAMI 버전을 사용하여 사용 가능한 지원되는 프레임워크 중 하나로 업그레이드하는 것이 좋습니다. 사용 가능한 DLAMIs 목록은 [릴리스 노트를 참조하십시오. DLAMI](#)

의 이전 버전을 DLAMIs 지원하나요 TensorFlow?

아니요. 에 명시된 대로 최초 GitHub 릴리스로부터 365일 동안 릴리스된 각 프레임워크의 최신 메이저 버전의 최신 패치 버전을 지원합니다. [AWS Deep Learning AMIs 프레임워크 지원 정책 표](#) 자세한 내

용은 [지원되는 프레임워크 표에 없는 버전을 실행 중인 경우에는 어떻게 해야 하나요?](#) 단원을 참조하세요.

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

a를 최신 프레임워크 버전과 DLAMI 함께 사용하려면 [DLAMIID](#)를 검색하고 이를 사용하여 [EC2콘솔을 DLAMI](#) 사용하여 실행하십시오. 예를 들어: AWS CLI 검색 명령 AWS Deep Learning AMIs ID는 DLAMI 릴리스 노트 페이지 [단일 프레임워크 DLAMI 릴리스](#) 노트를 참조하십시오. 선택한 프레임워크 버전은 [Supported] 라는 레이블이 붙어 있어야 합니다. [AWS Deep Learning AMIs 프레임워크 지원 정책 표](#)

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 버전을 모니터링합니다 (예: TensorFlow 2.9 ~ TensorFlow 2.9.1) 및 새로운 마이너 릴리스 버전 (예: TensorFlow TensorFlow 2.9~2.10), 그리고 최대한 빠른 시일 내에 사용할 수 있도록 합니다. 의 기존 버전이 새 버전과 함께 TensorFlow 출시되면 새 버전에 DLAMI 대한 지원과 TensorFlow 함께 해당 버전의 새 버전이 릴리스 됩니다. CUDA CUDA

워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요?

아니요. 에 대한 DLAMI 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?

릴리스 노트 페이지에서 이미지를 정기적으로 확인하세요. 패치가 적용되거나 업데이트된 새 프레임워크가 출시되면 해당 프레임워크로 업그레이드하는 것이 좋습니다. 사용 가능한 DLAMIs 목록은 [릴리스 노트를 참조하십시오. DLAMI](#)

프레임워크 버전을 변경하지 않고도 종속성이 업데이트되나요?

종속성은 프레임워크 버전을 변경하지 않고 업데이트됩니다. 하지만 종속성 업데이트로 인해 비호환성이 발생하는 경우 다른 버전으로 이미지를 생성합니다. [릴리스 노트에서](#) 업데이트된 종속성 정보를 확인하십시오. DLAMI

내 프레임워크 버전에 대한 능동 지원은 언제 종료되나요?

DLAMI 이미지는 변경할 수 없습니다. 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 end-of-support](#)

Note

버전 패치 업그레이드 및 보안 패치의 빈도가 높으므로 릴리스 노트 페이지를 DLAMI 자주 확인하고 변경 사항이 있을 때 업그레이드하는 것이 좋습니다.

프레임워크 버전(패치) 업그레이드

TensorFlow 2.7.0 기반 DLAMI 워크로드가 있고 버전 TensorFlow 2.7.1을 기반으로 하는 경우 GitHub AWS 2.7.1과 함께 새 버전이 릴리스됩니다. DLAMI TensorFlow 2.7.1이 적용된 새 이미지가 릴리스 되면 2.7.0이 적용된 TensorFlow 이전 이미지는 더 이상 활발하게 유지되지 않습니다. DLAMI with TensorFlow 2.7.0에는 추가 패치가 제공되지 않습니다. 그러면 TensorFlow 2.7의 DLAMI 릴리스 노트 페이지가 최신 정보로 업데이트됩니다. 각 마이너 패치에 대한 개별 릴리스 노트 페이지는 없습니다.

패치 업그레이드로 인해 새로 DLAMIs 생성된 항목은 새 [AMIID로](#) 지정됩니다.

AWS 보안 패치

TensorFlow 2.7.0 및 이미지 기반 워크로드가 있는 경우 AWS 보안 패치를 만들면 2.7.0용으로 TensorFlow 새 버전이 DLAMI 릴리스됩니다. TensorFlow 2.7.0이 설치된 이전 버전의 이미지는 더 이상 활발하게 유지 관리되지 않습니다. 자세한 내용은 최신 버전을 DLAMI 찾는 단계를 참조하십시오 [워크로드가 실행되는 동안 인스턴스가 제대로 패치되나요? 지원되는 프레임워크 버전에 대해 최신 패치가 적용된 이미지를 찾으려면 어떻게 해야 하나요?](#)

패치 업그레이드로 인해 새로 DLAMIs 생성된 항목은 새 [AMIID로](#) 지정됩니다.

패치 종료일(에이징 아웃)

DLAMIs GitHub 출시일로부터 365일 후에 패치 날짜가 종료됩니다.

[멀티 DLAMIs 프레임워크의](#) 경우 프레임워크 버전 중 하나가 업데이트되면 업데이트된 버전의 새 DLAMI 버전이 필요합니다. 이전 프레임워크 버전은 더 이상 활발하게 유지 관리되지 않습니다. DLAMI

Important

주요 프레임워크 업데이트가 있는 경우에는 예외가 적용됩니다. 예를 들어 TensorFlow 1.15가 TensorFlow 2.0으로 업데이트되면 GitHub 릴리스일로부터 2년 또는 원본 프레임워크 유지 관리 팀이 지원을 중단한 후 6개월 중 더 빠른 날짜에 해당하는 기간 동안 최신 버전의 TensorFlow 1.15를 계속 지원합니다.

종속성 end-of-support

Python 3.6이 설치된 TensorFlow 2.7.0 DLAMI 이미지에서 워크로드를 실행하고 있고 해당 버전의 Python이 표시되면 Python 3.6을 기반으로 하는 모든 DLAMI 이미지가 더 이상 활발하게 유지 관리되지 않습니다. end-of-support 마찬가지로 Ubuntu 16.04와 같은 OS 버전을 대상으로 end-of-support 표시하면 Ubuntu 16.04에 종속된 모든 DLAMI 이미지가 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 DLAMI a를 사용하려면 DLAMIID를 검색하고 이를 사용하여 콘솔을 사용하여 실행하십시오.](#) DLAMI EC2에 대해 AWS CLIAMIID를 검색하는 명령은 [단일 프레임워크 DLAMI 릴리스 노트의 릴리스 노트 페이지](#)를 참조하십시오.

프레임워크 및 해당 버전의 지원 변경 사항을 계속 up-to-date 확인하려면 어떻게 해야 합니까?

up-to-date 를 사용하여 DLAMI 프레임워크와 버전을 계속 사용할 수 있습니다. [AWS Deep Learning AMIs 프레임워크 지원 정책 표](#), [DLAMI 릴리스 노트](#)

Anaconda 리포지토리를 사용하려면 상용 라이선스가 필요한가요?

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

주요 NVIDIA 드라이버 변경 DLAMIs

2023년 11월 15일에 AWS 에 대한 중요한 변경 사항을 적용했습니다. AWS Deep Learning AMIs (DLAMI) DLAMIs 사용하는 NVIDIA 드라이버와 관련이 있습니다. 변경된 내용 및 사용에 영향을 미치는지 여부에 대한 자세한 내용은 [DLAMINVIDIA드라이버 변경 FAQs](#).

DLAMINVIDIA드라이버 변경 FAQs

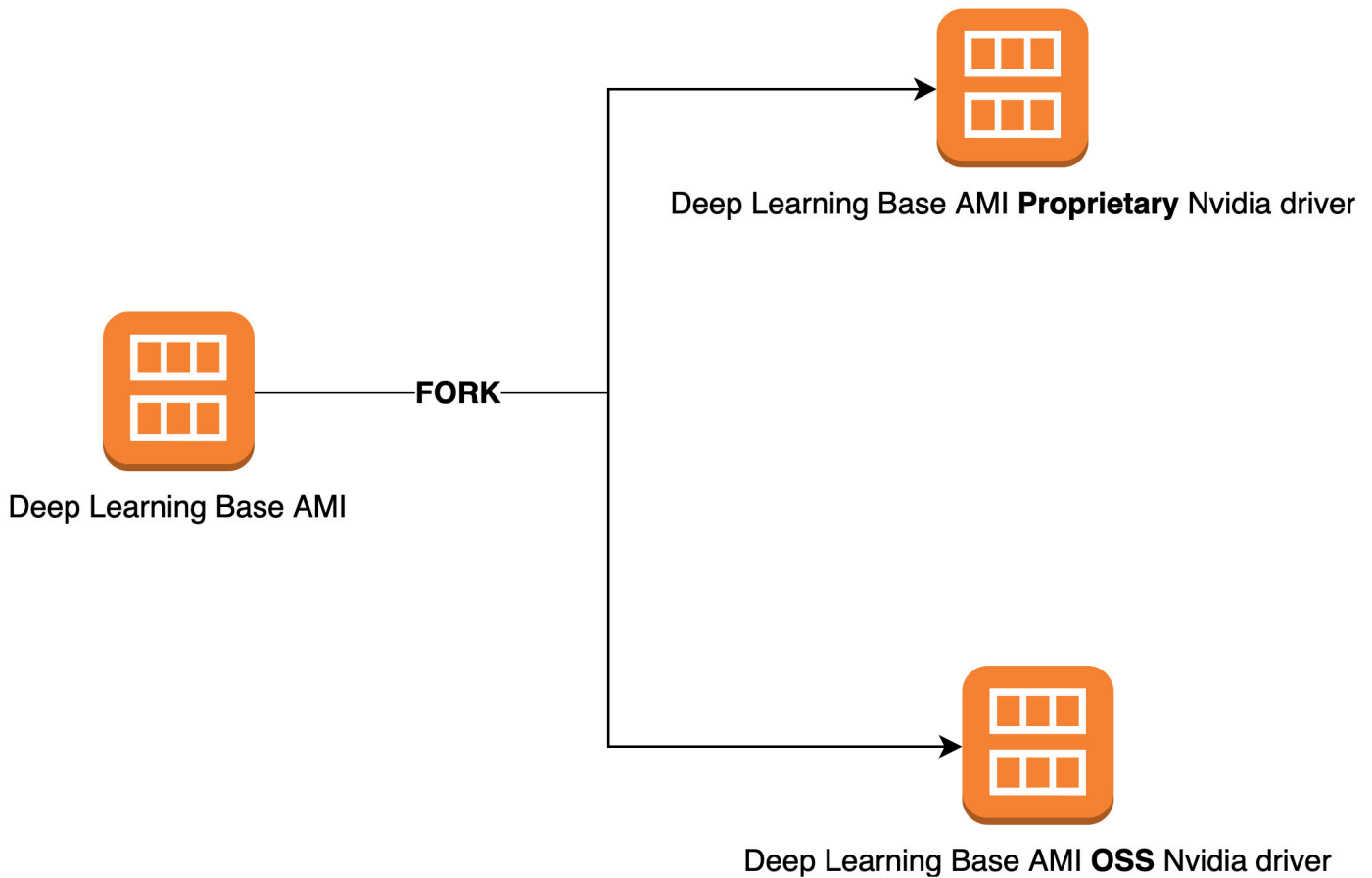
- [무엇이 바뀌었나요?](#)
- [이 변경이 필요한 이유는 무엇인가요?](#)
- [이번 변경은 어떤 영향을 미쳤습니까? DLAMIs](#)
- [이것이 여러분에게 어떤 의미가 있나요?](#)
- [최신 버전에서 기능 손실이 발생합니까? DLAMIs](#)
- [이 변경 사항이 Deep Learning Containers에 영향을 미쳤습니까?](#)

무엇이 바뀌었나요?

우리는 두 그룹으로 DLAMIs 나뉘었습니다.

- DLAMIsNVIDIA전용 드라이버 사용 (P3, P3dn, G3 지원)
- DLAMIs해당 NVIDIA OSS 드라이버 사용 (G4dn, G5, P4, P5 지원)

따라서 두 범주 DLAMIs 각각에 대해 새 이름과 새 이름을 사용하여 새 이름을 만들었습니다. AMI IDs DLAMIs이들은 서로 바뀌어서 사용할 수 없습니다. 즉, DLAMIs 한 그룹의 인스턴스는 다른 그룹에서 지원하는 인스턴스를 지원하지 않습니다. 예를 들어 P5를 DLAMI 지원하는 그룹은 G3을 지원하지 않고 G3를 DLAMI 지원하는 그룹은 P5를 지원하지 않습니다.



이 변경이 필요한 이유는 무엇인가요?

이전에는 DLAMIs의 전용 커널 드라이버가 NVIDIA GPUs 포함되어 있었습니다. NVIDIA 하지만 업스트림 Linux 커널 커뮤니티에서는 드라이버와 같은 전용 커널 드라이버가 다른 커널 드라이버와 통신하지 못하도록 NVIDIA GPU 하는 변경 사항을 수용했습니다. 이 변경으로 인해 분산 GPUDirect RDMA 교육에 효율적으로 사용할 수 있는 메커니즘인 P4 및 P5 시리즈 인스턴스에서는 GPUs 비활성화됩니다. EFA 따라서 DLAMIs 이제 NVIDIA 오픈 소스 드라이버와 연결되어 G4dn, G5, P4, P5를 지원하는 OpenRM 드라이버 (오픈 소스 EFA 드라이버) 를 사용하게 되었습니다. 하지만 이 OpenRM 드라이버는 이전 인스턴스 (예: P3 및 G3) 를 지원하지 않습니다. 따라서 두 인스턴스 유형을 모두 지원하는 최신 성능 및 보안을 DLAMIs 계속 제공하기 위해 OpenRM 드라이버 (G4dn, G5, P4, P5 지원) 를 사용하는 그룹과 이전 전용 드라이버 (P3, P3dn 및 G3를 지원) 를 사용하는 DLAMIs 그룹으로 나누었습니다.

이번 변경은 어떤 영향을 미쳤습니까? DLAMIs

이 변경은 모두에게 영향을 미쳤습니다 DLAMIs.

이것이 여러분에게 어떤 의미가 있나요?

지원되는 Amazon Elastic Compute Cloud (AmazonEC2) 인스턴스 유형에서 실행하는 한 모두 DLAMIs 기능, 성능 및 보안을 계속 제공합니다. a가 DLAMI 지원하는 EC2 인스턴스 유형을 확인하려면 해당 유형에 대한 릴리스 노트를 확인한 다음 지원되는 EC2 인스턴스를 찾아보십시오. DLAMI 현재 지원되는 DLAMI 옵션 목록과 해당 릴리스 노트 링크는 를 참조하십시오 [DLAMIs 릴리스 정보](#).

또한 올바른 방법을 사용해야 합니다. AWS Command Line Interface (AWS CLI) 명령을 사용하여 현재 DLAMIs 명령을 호출합니다.

P3, P3dn 및 G3을 DLAMIs 지원하는 베이스의 경우 다음 명령을 사용하십시오.

```
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, P5를 DLAMIs 지원하는 베이스의 경우 다음 명령을 사용하십시오.

```
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 버전에서 기능 손실이 발생합니까? DLAMIs

아니요, 기능 손실은 없습니다. 지원되는 EC2 인스턴스 유형에서 실행하는 경우 현재는 이전 버전과 같은 모든 기능 DLAMIs, 성능 및 보안을 DLAMIs 제공합니다.

이 변경 사항이 Deep Learning Containers에 영향을 미쳤습니까?

아니요, 이 변경 사항은 영향을 미치지 않았습니다. AWS Deep Learning Containers는 NVIDIA 드라이버를 포함하지 않기 때문입니다. 하지만 기본 인스턴스와 AMIs 호환되는 곳에서 Deep Learning Containers를 실행해야 합니다.

에 대한 관련 정보 DLAMI

DLAMI외부 관련 정보가 있는 다른 리소스를 찾을 수 있습니다. AWS Deep Learning AMIs 개발자 안내서 커집 AWS re:Post다른 고객의 질문을 확인하거나 직접 질문해 보세요. DLAMI 에 AWS Machine Learning 블로그 및 기타 AWS 블로그, 에 대한 DLAMI 공식 게시물을 읽어보세요.

AWS re:Post

[태그: AWS Deep Learning AMIs](#)

AWS 블로그

- [AWS Machine Learning 블로그 | 카테고리: AWS Deep Learning AMIs](#)
- [AWS Machine Learning 블로그 | Amazon EC2 C5 및 P3 인스턴스에 최적화된 TensorFlow 1.6을 통한 교육 속도 향상](#)
- [AWS Machine Learning 블로그 | 신규 AWS Deep Learning AMIs Machine Learning 실무자용](#)
- [AWS Partner Network \(APN\) 블로그 | 신규 교육 과정 이용 가능: Machine Learning 및 딥러닝 소개 AWS](#)
- [AWS 뉴스 블로그 | 딥 러닝을 통한 딥 러닝으로의 여정 AWS](#)

DLAMIs 릴리스 정보

여기에서 현재 지원되는 모든 항목에 대한 자세한 릴리스 노트를 찾을 수 있습니다. AWS Deep Learning AMIs (DLAMI) 옵션.

더 이상 지원하지 않는 DLAMI 프레임워크의 릴리스 노트는 Framework [DLAMISupport Policy](#) 페이지의 지원되지 않는 프레임워크 릴리스 노트 아카이브 섹션을 참조하십시오.

Note

The AWS Deep Learning AMIs 보안 패치는 매일 밤마다 릴리스해야 합니다. 공식 릴리스 노트에는 이러한 중분 보안 패치를 포함하지 않습니다.

베이스 DLAMIs

GPU

- X86
 - [AWS 딥 러닝 베이스 AMI \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 베이스 AMI \(우분투 22.04\)](#)
 - [AWS 딥 러닝 베이스 AMI \(우분투 20.04\)](#)
- ARM64
 - [AWS 딥 러닝 베이스 ARM64 AMI \(우분투 22.04\)](#)
 - [AWS 딥 러닝 베이스 ARM64 AMI \(아마존 리눅스 2\)](#)

AWS 뉴런

- X86
 - [AWS 딥 러닝 베이스 AMI 뉴런 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 베이스 AMI 뉴런 \(우분투 20.04\)](#)

퀵컴

- X86

- [AWS 딥러닝 베이스 쿨컴 \(AMI아마존 리눅스 2\)](#)

단일 프레임워크 DLAMIs

PyTorch-특정 AMIs

GPU

- X86
 - [AWS 딥 러닝 AMI GPU PyTorch 2.3 \(우분투 20.04\)](#)
 - [AWS 딥 러닝 AMI GPU PyTorch 2.3 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI GPU PyTorch 2.2 \(우분투 20.04\)](#)
 - [AWS 딥 러닝 AMI GPU PyTorch 2.2 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI GPU PyTorch 1.13 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI GPU PyTorch 1.13 \(우분투 20.04\)](#)
- ARM64
 - [AWS 딥 러닝 ARM64 AMI GPU PyTorch 2.3 \(우분투 22.04\)](#)
 - [AWS 딥 러닝 ARM64 AMI GPU PyTorch 2.2 \(우분투 20.04\)](#)

AWS 뉴런

- X86
 - [AWS 딥 러닝 AMI 뉴런 PyTorch 1.13 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI 뉴런 PyTorch 1.13 \(우분투 20.04\)](#)

TensorFlow-특정 AMIs

GPU

- X86
 - [AWS 딥 러닝 AMI GPU TensorFlow 2.16 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI GPU TensorFlow 2.16 \(우분투 20.04\)](#)
 - [AWS 딥 러닝 AMI GPU TensorFlow 2.15 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI GPU TensorFlow 2.15 \(우분투 20.04\)](#)

AWS 뉴런

- X86
 - [AWS 딥 러닝 AMI 뉴런 TensorFlow 2.10 \(아마존 리눅스 2\)](#)
 - [AWS 딥 러닝 AMI 뉴런 TensorFlow 2.10 \(우분투 20.04\)](#)

멀티 프레임워크 DLAMIs

Tip

기계 학습 프레임워크를 하나만 사용하는 경우 [단일 DLAMI](#) 프레임워크를 사용하는 것이 좋습니다.

GPU

- X86
 - [AWS 딥 러닝 AMI \(아마존 리눅스 2\)](#)

AWS 뉴런

- X86
 - [AWS 딥 러닝 AMI 뉴런 \(우분투 22.04\)](#)

더 이상 사용되지 않는 기능 DLAMI

다음 표에는 더 이상 사용되지 않는 기능이 나열되어 있습니다. AWS Deep Learning AMIs (DLAMI), 더 이상 사용되지 않는 날짜, 더 이상 사용되지 않는 이유에 대한 세부 정보.

기능	날짜	Details
Ubuntu 16.04	2021년 10월 7일	우분투 리눅스 LTS 16.04는 2021년 4월 30일에 5년의 서비스 LTS 기간이 종료되었으며 벤더에서 더 이상 지원하지 않습니다. 2021년 10월 현재 새 릴리스에서 딥 러닝 베이스 AMI (우분투 16.04)에 대한 업데이트는 더 이상 없습니다. 이전 릴리스는 계속 사용할 수 있습니다.
Amazon Linux	2021년 10월 7일	아마존 end-of-life 리눅스는 2020년 12월 기준입니다. 2021년 10월 현재 새 릴리스에는 딥 러닝 AMI (Amazon Linux)에 대한 업데이트가 더 이상 없습니다. 딥 러닝 AMI (Amazon Linux)의 이전 릴리스는 계속 사용할 수 있습니다.
Chainer	07/01/2020	Chainer는 2019년 12월부터 주요 릴리스의 종료 를 공지했습니다. 따라서 2020년 DLAMI 7월부터 더 이상 차이 너 콘다 환경을 포함하지 않을 예정입니다. 이러한 환경을 DLAMI 포함하는 이전 릴리스는 계속 사용할 수 있

기능	날짜	Details
		<p>습니다. 이러한 프레임워크에 대해 오픈 소스 커뮤니티에서 게시한 보안 수정 사항이 있는 경우에만 이러한 환경에 대한 업데이트를 제공합니다.</p>
Python 3.6	06/15/2020	<p>고객의 요청에 따라 새로운 TF/MX/PT 릴리스를 위해 Python 3.7로 이전하고 있습니다.</p>
Python 2	01/01/2020	<p>Python 오픈 소스 커뮤니티는 Python 2에 대한 지원을 공식적으로 종료했습니다.</p> <p>TensorFlow, PyTorch, 및 MXNet 커뮤니티에서도 TensorFlow 1.15, TensorFlow 2.1, PyTorch 1.4, MXNet 1.6.0 릴리스가 Python 2를 지원하는 마지막 릴리스가 될 것이라고 발표했습니다.</p>

DLAMI에 대한 문서 기록

다음 표에는 최신 DLAMI 릴리스 기록 및 관련 변경 사항이 나와 있습니다. AWS Deep Learning AMIs 개발자 안내서

최근 변경

변경 사항	설명	날짜
ARM64 DLAMI	더 AWS Deep Learning AMIs 이제 Arm64 프로세서 기반 GPUs 이미지를 지원합니다.	2021년 11월 29일
TensorFlow 2	Conda를 AMI 사용한 딥 러닝은 이제 TensorFlow 2와 CUDA 10이 함께 제공됩니다.	2019년 12월 3일
AWS 인퍼런시아	딥러닝은 이제 다음을 지원합니다. AWS 인퍼런시아 하드웨어 및 AWS 뉴런. SDK	2019년 12월 3일
TensorFlow 인셉션 모델과 함께 서빙 사용하기	Elastic Inference가 있는 경우와 사용하지 않는 경우 모두 Servicing에 TensorFlow Inception 모델과 함께 추론을 사용하는 예제가 추가되었습니다.	2018년 11월 28일
Elastic Inference	설정 가이드에 Elastic Inference 전제 조건 및 관련 정보가 추가되었습니다.	2018년 11월 28일
야간 빌드에서 설치하기 PyTorch	Conda를 사용하여 딥 러닝에서 야간 빌드를 제거한 다음 설치하는 방법을 설명하는 자습서가 추가되었습니다. PyTorch PyTorch AMI	2018년 9월 25일

[Conda 자습서](#)

예제는 최신 릴리스를 반영하도록 MOTD 업데이트되었습니다.

2018년 7월 23일

이전 변경사항

다음 표에는 2018년 7월 이전의 이전 DLAMI 릴리스 및 관련 변경 내역이 나와 있습니다.

변경 사항	설명	날짜
TensorFlow 호로보드와 함께	Horovod와 ImageNet 함께 TensorFlow 훈련할 수 있는 튜토리얼이 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 “딥 러닝 시작하기”라는 제목의 10분 분량의 튜토리얼의 첫 번째 릴리스입니다. AMI	2018년 4월 26일
Chainer 자습서	Chainer를 멀티 GPU GPU, 싱글 및 CPU 모드에서 사용하기 위한 튜토리얼이 추가되었습니다. CUDA 여러 프레임워크의 통합이 CUDA 8에서 CUDA 9로 업그레이드되었습니다.	2018년 2월 28일
Linux AMIs v3.0 및 MXNet 모델 서버 소개, TensorFlow 서비스 및 TensorBoard	모델 서버 v0.1.5, TensorFlow 서빙 v1.4.0 및 v0.4.0을 AMIs 사용한 새로운 MXNet 모델 및 시각화 제공 기능이 포함된 Conda용 자습서가 추가되	2018년 1월 25일

변경 사항	설명	날짜
	<p>있습니다. TensorBoard AMI 및 프레임워크 기능은 Conda 및 개요에 설명되어 있습니다. CUDA CUDA 최신 출시 정보를 https://aws.amazon.com/regions/leasenotes/(으)로 이동</p>	
리눅스 v2.0 AMIs	<p>베이스, 소스, 콘다가 2.1로 AMIs NCCL 업데이트되었습니다. 소스와 콘다는 MXNet v1.0, PyTorch 0.3.0, 케라스 2.0.9로 AMIs 업데이트되었습니다.</p>	2017년 12월 11일
AMI 두 개의 윈도우 옵션이 추가되었습니다.	<p>Windows 2012 R2 및 2016 AMIs 출시: AMI 선택 가이드에 추가되고 릴리스 노트에 추가되었습니다.</p>	2017년 11월 30일
최초 문서 릴리스	<p>변경된 주제/단원의 링크를 포함하는 변경 사항에 대한 자세한 설명.</p>	2017년 11월 15일

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