



開發人員指南

深度學習 AMI



深度學習 AMI: 開發人員指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 AWS Deep Learning AMI ?	1
關於本指南	1
先決條件	1
範例使用方式	1
功能	2
預先安裝架構	2
預先安裝的 GPU 軟體	2
模型服務和視覺化	2
入門	4
如何開始使用 DLAMI	4
DLAMI 選擇	4
CUDA 安裝和架構連結	5
基礎	6
Conda	6
架構	8
作業系統	8
執行個體選擇	8
定價	10
區域可用性	10
GPU	10
CPU	11
推論	12
草屬	12
哈瓦那	13
框架 Support 政策	14
支援的架構	14
常見問答集	14
哪些框架版本可以獲得安全補丁？	15
AWS發布新框架版本時會發布哪些圖像？	15
哪些圖像獲得新的 SageMaker/AWS功能？	15
如何在支持的框架表中定義當前版本？	15
如果我正在運行的版本不在支持框架表中怎麼辦？	15
DLAMI 是否支持以前版本的？ TensorFlow	16
如何找到支持的框架版本的最新補丁圖像？	16

新影像的發行頻率是多久？	16
當我的工作負載執行時，我的執行個體是否會被修補？	16
當新的修補或更新的框架版本可用時會發生什麼情況？	16
依賴關係是否更新而不更改框架版本？	16
我的框架版本的主動支持何時結束？	16
具有不再積極維護的框架版本的圖像是否會被修補？	18
如何使用較舊的框架版本？	18
如何保持 up-to-date 框架及其版本的支持更改？	18
我是否需要商業授權才能使用 Anaconda 軟體庫？	18
推出 DLAMI	19
步驟 1：啟動 DLAMI	19
擷取密碼 DLAMI 碼	20
從 Amazon EC2 主控台啟動	21
步驟 2：Connect 至 DLAMI	22
步驟 3：測試您的 DLAMI	22
步驟 4：管理您的 DLAMI 實例	22
清除	23
Jupyter 設定	23
保護 Jupyter	24
啟動伺服器	25
設定用戶端	25
登入 Jupyter 筆記本伺服器	27
使用 DLAMI	29
康達·DLAMI	29
使用康達介紹深度學習 AMI	29
登錄到您的 DLAMI	30
啟動 TensorFlow 環境	31
切換到 PyTorch Python 環境	32
切換至 MXNet 3 環境	33
移除環境	34
基地 DLAMI	34
使用深度學習基礎 AMI	34
設定 CUDA 版本	34
Jupyter 筆記本	34
瀏覽安裝教學課程	35
使用 Jupyter 切換環境	36

教學課程	36
10 分鐘教學課程	37
啟用架構	37
偵錯和視覺化	55
分散式訓練	60
Elastic Fabric Adapter	82
GPU 監控和最佳化	96
AWS 推論	105
重力子 DLAMI	126
哈瓦那·DLAMI	136
Inference	138
使用架構與 ONNX	143
模型服務	155
升級您的 DLAMI	163
DLAMI 升級	163
軟體更新	164
安全	165
資料保護	165
身分和存取權管理	166
使用身分來驗證	166
使用政策管理存取權	169
IAM 搭配 Amazon EMR	170
記錄和監控	171
使用量追蹤	171
合規驗證	171
恢復能力	172
基礎設施安全性	172
DLAMI 的重要變化	173
常見問答集	173
有什麼變化？	173
為什麼需要進行此更改？	174
哪些 DLAM 會受到此變更的影響？	174
這對你意味著什麼？	175
你應該什麼時候開始使用新的 DLAMI？	176
新的 DLAMI 會否有任何功能損失？	176
關於可攜式資料中心呢？	176

相關資訊	177
論壇	177
部落格	177
常見問答集	177
特拉米的發行公告	181
.....	181
基地 DLAMI	181
單框架 DLAMI	181
多框架	182
DLAMI 的通知	183
文件歷史記錄	185
AWS 詞彙表	189
.....	CXC

什麼是 AWS Deep Learning AMI ?

歡迎使用 AWS Deep Learning AMI 使用者指南，

AWS Deep Learning AMI(DLAMI) 是您在雲端進行深度學習的一站式商店。這個自訂機器執行個體可在大多數 Amazon EC2 區域使用，適用於各種執行個體類型，從僅 CPU 的小型執行個體到最新的高功能多 GPU 執行個體。它已預先配置 [NVIDIA CUDA](#) 和 [NVIDIA CUDNN](#)，以及最受歡迎的深度學習架構的最新版本。

關於本指南

本指南將幫助您啟動和使用 DLAMI。本指南涵蓋深度學習訓練和推論中幾個常見的使用案例。也涵蓋如何選擇適合您用途的正確 AMI，以及您可能偏好的執行個體類型。DLAMI 為每個框架提供了幾個教程。它還提供有關分佈式培訓，調試，使用推論和其他關鍵概念的教程。您可以找到如何在瀏覽器中設定 Jupyter 以執行教學課程的說明。

先決條件

您應該熟悉命令行工具和基本 Python 才能成功運行 DLAMI。架構本身會提供如何使用每個架構的教學課程，不過，本指南也會說明如何啟用每個架構，以及如何找到適當的入門教學課程。

DLAMI 的範例

深度學習：DLAMI 是學習或教授機器學習和深度學習架構的絕佳選擇。它讓您無須花心力對每個架構的安裝進行疑難排解，也無須費神讓這些架構在相同電腦上共同運作。DLAMI 隨附 Jupyter 筆記本，可讓您輕鬆為剛接觸機器學習和深度學習的人員執行架構提供的教學課程。

應用程序開發：如果您是應用程序開發人員，並且有興趣使您的應用程序利用 AI 的最新進展，那麼 DLAMI 是您的理想測試平台。每個架構都隨附如何開始使用深度學習的教學課程，其中多數提供 Model Zoo，讓您無需自行建立神經網路或進行任何模型訓練，就能輕鬆試用深度學習。有些範例會說明如何在幾分鐘內建置影像偵測應用程式，或是如何為您自己的聊天機器人建置語音辨識應用程式。

機器學習和資料分析：如果您是資料科學家或有興趣使用深度學習來處理您的資料，您會發現許多架構支援 R 和 Spark。您可以找到如何執行簡單迴歸的教學課程，一路到如何為個人化和預測系統建置可擴展性資料處理系統的教學課程。

研究：如果您是研究人員，並且想要嘗試新框架，測試新模型或訓練新模型，則 DLAMI 和擴展 AWS 功能可以減輕繁瑣安裝和管理多個培訓節點的痛苦。

Note

雖然您最初的選擇可能是將執行個體類型升級為具有更多 GPU (最多 8 個) 的較大執行個體，但您也可以建立 DLAMI 執行個體叢集，以水平擴展。如需叢集建置的詳細資訊，請查看[相關資訊](#)。

DLAMI Machine 的功能

預先安裝架構

目前 DLAMI 有兩種主要風格，以及與操作系統 (OS) 和軟件版本相關的其他變化：

- [使用康達深度學習 AMI](#) - 使用 conda 套件和單獨 Python 環境，分別安裝架構
- [深度學習基礎 AMI](#)-沒有安裝框架; 只有 [NVIDIA CUDA](#) 和其他依賴關係

具有 Conda 的深度學習 AMI 使用 conda 環境隔離每個框架，因此您可以隨意在它們之間切換，而不必擔心它們的依賴關係發生衝突。

以下是與 Conda 搭配的深度學習 AMI 支援架構的完整清單：

- 阿帕奇 MXnet (孵化中)
- PyTorch
- TensorFlow 2

Note

我們不再包括 CNTK，咖啡，Caffe2，西亞諾，查納或凱拉斯康達環境在 v28 版本 AWS Deep Learning AMI 開始。包含這些環境的舊版 AWS Deep Learning AMI 將可繼續使用。不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

預先安裝的 GPU 軟體

即使您使用僅限 CPU 的執行個體，DLAMI 也會擁有 [NVIDIA CUDA](#) 和 [NVIDIA 固定 cuDNN](#) 置。無論執行個體類型為何，安裝的軟體都相同。請記住，GPU 特定工具只適用於至少有一個 GPU 的執行個體。[選取 DLAMI 的例證類型](#) 中提供更多詳細資訊。

如需 CUDA 安裝的詳細資訊，請參閱[CUDA 安裝和架構連結](#)。

模型服務和視覺化

具有 Conda 的深度學習 AMI 預先安裝了兩種模型伺服器，一種用於 MXNet TensorFlow，另一種用於模型視覺化。TensorBoard

- [Apache MXNet 專用模型伺服器 \(MMS\)](#)
- [TensorFlow 服務](#)
- [TensorBoard](#)

入門

如何開始使用 DLAMI

本指南包含如何挑選適合您的 DLAMI、選取適合您使用案例和預算的執行個體類型，以及[相關資訊](#)說明可能感興趣的自訂設定的秘訣。

如果您不熟悉使用AWS或使用 Amazon EC2，請從[使用康達深度學習 AMI](#)。如果您熟悉 Amazon EC2 和 Amazon EMR、Amazon EFS 或 Amazon S3 等其他AWS服務，並且對於需要分散式訓練或推論的專案整合這些服務感興趣，請查[相關資訊](#)看是否適合您的使用案例。

我們建議您先查看[選擇您的 DLAMI](#)，了解哪些執行個體類型可能最適合您的應用程式。

另一個選擇是這個快速教程：[啟動一個AWS Deep Learning AMI \(在 10 分鐘內 \)](#)。

後續步驟

[選擇您的 DLAMI](#)

選擇您的 DLAMI

我們提供一系列的 DLAMI 選項。為了協助您針對使用案例選擇正確的 DLAMI，我們會依照開發映像的硬體類型或功能來分組映像。我們的頂級分組是：

- DLAMI 類型：[CUDA](#) 與[基礎與單框架與多框架](#) (康達 DLAMI)
- 運算架構：[x86](#) 型與 [ARM](#) 為基礎的[AWS重力子](#)
- 處理器類型：[GPU](#) 與 [CPU](#) 與[推論與哈瓦那](#)
- SDK：[CUDA](#) 與[AWS神經元與突觸](#)
- 操作系統：[亞馬遜 Linux](#) 與 [Ubuntu](#)

本指南中的其餘主題有助於進一步通知您並進入更多詳細信息。

主題

- [CUDA 安裝和架構連結](#)
- [深度學習基礎 AMI](#)

- [使用康達深度學習 AMI](#)
- [CPU 架構選項](#)
- [DLAMI 作業系統選項](#)

接下來

[使用康達深度學習 AMI](#)

CUDA 安裝和架構連結

雖然深度學習都是相當前沿的，但每個框架都提供了「穩定」的版本。這些穩定版本可能無法使用最新的 CUDA 或 cuDNN 實作和功能。您的使用案例和所需的功能可協助您選擇架構。如果您不確定，請搭配 Conda 使用最新的深度學習 AMI。它具有 CUDA 10 的所有框架的官方 pip 二進製文件，使用每個框架支持的最新版本。如果您想要最新版本，並自訂您的深度學習環境，請使用深度學習基礎 AMI。

請查看[穩定版與發行候選](#)上的指南，以獲得進一步指導。

選擇帶有 CUDA 的 DLAMI

[深度學習基礎 AMI](#)擁有所有可用的 CUDA 11 系列，包括 11.0、11.1 和 11.2。

[使用康達深度學習 AMI](#)擁有所有可用的 CUDA 11 系列，包括 11.0、11.1 和 11.2。

Note

我們不再包括 CNTK，咖啡，Caffe2，西亞諾，查納或凱拉斯康達環境在 v28 版本 AWS Deep Learning AMI 開始。包含 AWS Deep Learning AMI 這些環境的舊版繼續可用。不過，我們只會在開放原始碼社群針對這些架構發佈安全性修正時，才會提供這些環境的更新。

如需特定架構版本號碼，請參閱[特拉米的發行公告](#)

選擇此 DLAMI 類型，或使用「下一步」選項了解有關不同 DLAMI 的更多信息。

選擇其中一個 CUDA 版本，並查看附錄中具有該版本的 DLAMI 的完整列表，或使用「下一步」選項了解有關不同 DLAMI 的更多信息。

接下來

[深度學習基礎 AMI](#)

相關主題

- 如需在 CUDA 版本間切換的說明，請參閱 [使用深度學習基礎 AMI 教學課程](#)。

深度學習基礎 AMI

深度學習基礎 AMI 就像是用於深度學習的空白畫布。它配備了您需要的一切，直到安裝特定框架的點，並可以選擇 CUDA 版本。

為什麼選擇基地 DLAMI

這個 AMI 群組適合希望延伸深度學習專案和建置最新版本的專案參與者。也適合想開發自己環境的人，並確信最新的 NVIDIA 軟體已安裝和運作，讓他們可以專注在選擇他們要安裝的架構和版本。

選擇此 DLAMI 類型，或使用「下一步」選項了解有關不同 DLAMI 的更多信息。

接下來

[DLAMI 與康達](#)

相關主題

- [使用深度學習基礎 AMI](#)

使用康達深度學習 AMI

康達 DLAMI 使用conda虛擬環境。這些環境的設定是為了讓不同的架構安裝分開，並簡化架構之間的切換作業。這對於學習和嘗試 DLAMI 必須提供的所有框架非常有用。大多數用戶發現，帶有 Conda 的新深度學習 AMI 非常適合他們。

這些 AMI 是主要的 DLAMI。它們經常使用框架的最新版本進行更新，並具有最新的 GPU 驅動程序和軟件。它們通常被稱為 [AWS Deep Learning AMI](#) 在大多數文件中。

- Ubuntu 18.04 DLAMI 具有以下框架：阿帕奇 MXNet (孵化) PyTorch，和 TensorFlow 2。
- 亞馬遜 Linux 2 DLAMI 具有以下框架：阿帕奇 MXNet (孵化) PyTorch，和 TensorFlow 2。

Note

從 v28 版本AWS Deep Learning AMI開始，我們不再包括 CNTK，咖啡，Caffe2，西亞諾，查納和凱拉斯康達環境。包含AWS Deep Learning AMI這些環境的舊版繼續可用。不過，我們只會在開放原始碼社群針對這些架構發佈安全性修正時，才會提供這些環境的更新。

穩定版與發行候選

Conda AMI 使用每個架構最新正式版本的最佳化二進位程式碼。不預期使用版本候選項目和實驗性功能。最佳化取決於架構對 Intel MKL DNN 等加速技術的支援，可加速 C5 和 C4 CPU 執行個體類型的訓練和推論。這些二進製文件也被編譯為支持高級英特爾指令集，包括但不限於 AVX，AVX-2，SSE4.1 和 SSE4.2。這些項目可加速 Intel CPU 架構上的向量和浮點操作。此外，對於 GPU 執行個體類型，CUDA 和 cuDNN 會以最新官方發行版本支援的任何版本進行更新。

搭配 Conda 的深度學習 AMI 會在架構首次啟用時，自動為您的 Amazon EC2 執行個體安裝最佳化的架構版本。如需詳細資訊，請參閱[搭配康達使用深度學習 AMI](#)。

如果您想使用自定義或優化構建選項從源代碼安裝，那麼[深度學習基礎 AMI](#) s 可能是您更好的選擇。

Python 2 棄用

Python 開放原始碼社群已於 2020 年 1 月 1 日正式終止支援 Python 2。TensorFlow 和 PyTorch 社區已經宣布 TensorFlow 2.1 和 PyTorch 1.4 版本是最後一個支持 Python 2 的版本。包含 Python 2 Conda 環境的舊版本 (v26、v25 等) 仍然可供使用。不過，只有在開放原始碼社群針對這些版本發佈的安全性修正程式時，我們才會在先前發佈的 DLAMI 版本上提供 Python 2 Conda 環境的更新。具有最新版本 TensorFlow 和 PyTorch 框架的 DLAMI 發行版本不包含 Python 2 Conda 環境。

CUDA 支援

您可以在 [GPU DLAMI 版本說明](#) 中找到特定的 CUDA 版本號碼。

接下來

[CPU 架構選項](#)

相關主題

- 如需搭配 Conda 使用深度學習 AMI 的教學課程，請參閱[搭配康達使用深度學習 AMI](#)教學課程。

CPU 架構選項

AWS Deep Learning AMI提供 x86 架構或以 ARM 為基礎的[AWS重力 on2](#) CPU 架構。

選擇其中一個引力子 GPU DLAMI 來搭配 ARM 式 CPU 架構使用。所有其他 GPU 網路介面卡目前都是以 x86 為基礎的。

- [AWS深度學習 AMI 引力式 GPU 顯示卡 11.4 \(Ubuntu 20.04\)](#)
- [AWS深度 AMI 習引力式網路 GPU TensorFlow 2.6](#)
- [AWS深 AMI 學習引力式 GPU PyTorch 1.10](#)

如需有關重力子 GPU DLAMI 開始使用的資訊，請參閱[重力子 DLAMI](#)。如需可用執行個體類型的詳細資訊，請參閱[選取 DLAMI 的例證類型](#)。

接下來

[DLAMI 作業系統選項](#)

DLAMI 作業系統選項

DLAMI 在以下任一作業系統提供。

- Amazon Linux 2
- Ubuntu 20.04
- Ubuntu 18.04

舊版作業系統可在已淘汰的 DLAMI 上使用。有關 DLAMI 棄用的更多信息，請參閱 [DLAMI 的棄用](#)

在選擇 DLAMI 之前，請先評估您需要的執行個體類型，並識別您的AWS區域。

接下來

[選取 DLAMI 的例證類型](#)

選取 DLAMI 的例證類型

如需與特定 DLAMI 相容的推薦 Amazon EC2 執行個體系列，請參閱[AWS Deep Learning AMI目錄](#)。

更一般地說，在為 DLAMI 選取例證類型時，請考慮下列事項。

- 如果您是深度學習的新手，那麼具有單一 GPU 的執行個體可能會符合您的需求。
- 如果您注重預算，則可以使用僅限 CPU 的執行個體。
- 如果您想要針對深度學習模型推論最佳化高效能和成本效益，則可以搭配 AWS Inferentia 晶片使用執行個體。
- 如果您想要針對深度學習模型訓練最佳化高效能和成本效益，則可以搭配 Habana 加速器使用執行個體。
- 如果您正在尋找具有基於 ARM 的 CPU 架構的高性能 GPU 實例，則可以使用 G5G 實例類型。
- 如果您有興趣執行預先訓練的推論和預測模型，可以將 Amazon E [lastic Inference 附加到 Amazon EC2](#) 執行個體。Amazon Elastic Inference 可讓您使用 GPU 的一小部分存取加速器。
- 對於大量推論服務而言，具有大量記憶體之單一 CPU 執行個體或這類執行個體之叢集可能是更好的解決方案。
- 如果您使用的是具有大量資料或批次大小較大的大型模型，則需要具有更多記憶體之大型執行個體。您也可以將模型分發到 GPU 叢集。您可能會發現，如果您減少了批次大小，使用具有較少記憶體之執行個體會是更好的解決方案。這可能會影響準確度和訓練速度。
- 如果您有興趣使用 NVIDIA 集體通訊程式庫 (NCCL) 執行需要高層級節點間通訊的機器學習應用程式，您可能需要使用 E [lastic Fabric Adapter \(EFA\)](#)。

如需執行個體的詳細資訊，請參閱 [EC2 執行個體類型](#)

下列主題提供執行個體類型考量的資訊。

Important

深度學習 AMI 包括由 NVIDIA Corporation 開發、擁有或提供的驅動程式、軟體或工具組。您同意只在包含 NVIDIA 硬體的 Amazon EC2 執行個體上使用這些 NVIDIA 驅動程式、軟體或工具組。

主題

- [特殊護理 DLAMI 價](#)
- [DLAMI 區域供應情形](#)
- [建議的 GPU 執行個體](#)

- [建議的 CPU 執行個體](#)
- [建議的 Inference 執行個體](#)
- [建議的 Tference 執行個體](#)
- [Habana 執行個體](#)

特殊護理 DLAMI 價

DLAMI 中包含的深度學習框架是免費的，每個框架都有自己的開源許可證。雖然 DLAMI 中包含的軟體是免費的，但您仍然需要為基礎的 Amazon EC2 執行個體硬體付費。

部分 Amazon EC2 執行個體類型標示為免費。您可以在其中一個免費執行個體上執行 DLAMI。這意味著當您僅使用該實例的容量時，使用 DLAMI 是完全免費的。如果您需要具有更多 CPU 核心、更多磁碟空間、更多 RAM 或一或多個 GPU 的功能更強大的執行個體，則需要不在免費層執行個體類別中的執行個體。

如需執行個體選擇和定價的詳細資訊，請參閱 [Amazon EC2 定價](#)。

DLAMI 區域供應情形

每個區域都支援不同範圍的執行個體類型，而且執行個體類型在不同區域的成本通常會略有不同。並非每個區域都可以使用 DLAMI，但您可以將 DLAMI 複製到您選擇的區域。如需詳細資訊，[請參閱複製 AMI](#)。請記下「區域」選取清單，並確定您挑選的區域距離您或您的客戶很近。如果您打算使用多個 DLAMI 並可能建立叢集，請務必對叢集中的所有節點使用相同的區域。

如需區域的詳細資訊，請瀏覽 [EC2 區域](#)。

接下來

[建議的 GPU 執行個體](#)

建議的 GPU 執行個體

我們建議您使用 GPU 執行個體，用於大部分的深度學習在 GPU 執行個體上訓練新模型的速度比 CPU 執行個體更快。當您有多重 GPU 執行個體或想跨多個具 GPU 執行個體使用分散式訓練時，可以用子線性方式擴展。若要設定分散式訓練，請參閱 [分散式訓練](#)。

下列的執行個體類型支援 DLAMI。如需 GPU 執行個體類型選項及其用途的詳細資訊，請參閱 [EC2 執行個體類型](#) 並選取加速運算。

Note

模型大小應為選取執行個體時所需考量的因素。如果模型超過執行個體的可用 RAM，請選取具有適用於應用程式的足夠記憶體的不同執行個體類型。

- [Amazon EC2 P3 執行個體](#)擁有多達 8 個 NVIDIA 特斯拉 V100 GPU。
- [Amazon EC2 P4 實例](#)最多有 8 個 NVIDIA 特斯拉 A100 GPU。
- [Amazon EC2 G3 實例](#)最多有 4 個 NVIDIA 特斯拉 M60 GPU。
- [Amazon EC2 G4 執行個體](#)擁有高達 4 個 NVIDIA T4 GPU。
- [Amazon EC2 G5 執行個體](#)擁有高達 8 個 NVIDIA A10G GPU。
- [Amazon EC2 G5G 執行個體具有以 ARM 為基礎AWS的重力 2 處理器。](#)

DLAMI 執行個體提供用於監控和最佳化 GPU 程序的工具。如需監控 GPU 程序的詳細資訊，請參閱[GPU 監控和最佳化](#)。

如需使用 G5G 執行個體的特定教學課程，請參閱[重力子 DLAMI](#)。

接下來

[建議的 CPU 執行個體](#)

建議的 CPU 執行個體

無論您是預算有限、正在了解深度學習，或是只想要執行預測服務，CPU 類別都提供許多價格合理的選項。有些架構會利用 Intel 的 MKL DNN，加速 C5 (並非所有區域皆提供)、C4 和 C3 CPU 執行個體類型的訓練與推論。如需 CPU 執行個體類型的相關資訊，請參閱 [EC2 執行個體類型](#) 並選取運算優化

Note

模型大小應為選取執行個體時所需考量的因素。如果模型超過執行個體的可用 RAM，請選取具有適用於應用程式的足夠記憶體的不同執行個體類型。

- [Amazon EC2 執行個體擁有高達 72 個英特爾 vCPU](#) Us。C5 執行個體在科學建模、批次處理、分散式分析、高效能運算 (HPC) 以及機器和深度學習推論方面表現卓越。
- Amazon EC2 C4 執行個體擁有高達 36 個 Intel vCPU。

接下來

[建議的 Inferen 執行個體](#)

建議的 Inferen 執行個體

AWS Inferen 執行個體旨在提供高效能和具成本效益的深度學習模型 Inferen 工作負載。具體來說，Inf2 執行個體類型使用 AWS 推論晶片和 [AWS Neuron SDK](#)，這與熱門的機器學習架構 (例如和) 整合。TensorFlow PyTorch

客戶可以使用 Inf2 執行個體，以最低的成本在雲端執行大規模的機器學習推論應用程式，例如搜尋、推薦引擎、電腦視覺、語音辨識、自然語言處理、個人化和詐騙偵測。

Note

模型大小應為選取執行個體時所需考量的因素。如果模型超過執行個體的可用 RAM，請選取具有適用於應用程式的足夠記憶體的不同執行個體類型。

- [Amazon EC2 Inf2 執行個體](#) 具有高達 16 個 AWS 推論晶片和 100 Gbps 的聯網輸送量。

如需有關 AWS Inferen DLI 入門的詳細資訊，請參閱 [具有 DL AWS AMI 的推論晶片](#)

接下來

[建議的 Tferen 執行個體](#)

建議的 Tferen 執行個體

AWS Trainium 執行個體旨在提供高效能和具成本效益的深度學習模型 Inferen 工作負載。具體來說，Trn1 執行個體類型使用 AWS Trainium 晶片和 [AWS Neuron SDK](#)，這與熱門的機器學習架構 (例如和) 整合。TensorFlow PyTorch

客戶可以使用 Trn1 執行個體，在雲端中以最低的成本執行大規模的機器學習推論應用程式，例如搜尋、推薦引擎、電腦視覺、語音辨識、自然語言處理、個人化和詐騙偵測。

Note

模型大小應為選取執行個體時所需考量的因素。如果模型超過執行個體的可用 RAM，請選取具有適用於應用程式的足夠記憶體的不同執行個體類型。

- [Amazon EC2 Trn1 執行個體](#)具有高達 16 個AWS晶片和 100 Gbps 的聯網輸送量。

接下來

[Habana 執行個體](#)

Habana 執行個體

具有 Habana 加速器的執行個體旨在提供高效能和具成本效益的深度學習模型訓練工作負載。具體來說，DL1 執行個體類型使用 Intel 公司 Habana 實驗室的加速器。DL1 執行個體非常適合訓練機器學習模型，例如自然語言處理、物件偵測與分類、建議引擎，以及自駕車感知。

具有 Habana 加速器的執行個體配置了 Habana SynapseAI 軟體，並預先與流行的機器學習框架（例如和）整合。TensorFlow PyTorch如果您正在尋找效能和價格的最佳組合來訓練深度學習模型，請考慮使用 Habana 加速器的執行個體，以最低的訓練成本。

Note

模型大小應為選取執行個體時所需考量的因素。如果模型超過執行個體的可用 RAM，請選取具有適用於應用程式的足夠記憶體的不同執行個體類型。

- [Amazon EC2 DL1 執行個體](#)具有多達八個哈瓦那高迪加速器、256GB 的加速器記憶體、4 TB 的本機 NVMe 儲存，以及 400 Gbps 的聯網輸送量。

如需有關 Habana D1ami 入門的詳細資訊，請參閱。[哈瓦那 DLAMI](#)

框架 Support 政策

[AWS Deep Learning AMIs \(DLAMI\)](#) 可簡化深度學習工作負載的映像組態，並透過最新的架構、硬體、驅動程式、程式庫和作業系統進行最佳化。本頁詳細介紹了 DLAMI 的框架支持策略。如需可用 DLAMI 的清單，請參閱 DLAMI 的[版本說明](#)。

支援的架構

請參考下列[AWS Deep Learning AMI 架構 Support 援原則表格](#)，以檢查主動支援的架構和版本。

請參閱修補程式結束，瞭解原始架構維護團隊主動 AWS 支援的目前版本支援的時間長度。架構和版本可在單一架構 DLAMI 或多架構 DLAMI 中使用。

Note

在框架版本 x.y.z 中，x 指的是主要版本，y 指的是次要版本，z 指的是補丁版本。例如，對於 TensorFlow 2.6.5，主要版本是 2，次要版本為 6，修補程式版本為 5。

如需特定影像的詳細資訊，請參閱版本說明：

- [單一架構 DLAMI 發行說明](#)
- [多架構 DLAMI 發行說明](#)

常見問答集

- [哪些框架版本可以獲得安全補丁？](#)
- [AWS 發布新框架版本時會發布哪些圖像？](#)
- [哪些圖像獲得新的 SageMaker/AWS 功能？](#)
- [如何在支持的框架表中定義當前版本？](#)
- [如果我正在運行的版本不在支持框架表中怎麼辦？](#)
- [DLAMI 是否支持以前版本的 TensorFlow？](#)
- [如何找到支持的框架版本的最新補丁圖像？](#)
- [新影像的發行頻率是多久？](#)

- [當我的工作負載執行時，我的執行個體是否會被修補？](#)
- [當新的修補或更新的框架版本可用時會發生什麼情況？](#)
- [依賴關係是否更新而不更改框架版本？](#)
- [我的框架版本的主動支持何時結束？](#)
- [具有不再積極維護的框架版本的圖像是否會被修補？](#)
- [如何使用較舊的框架版本？](#)
- [如何保持 up-to-date 框架及其版本的支持更改？](#)
- [我是否需要商業授權才能使用 Anaconda 軟體庫？](#)

哪些框架版本可以獲得安全補丁？

如果架構版本在「[架構支援政策](#)」表格中AWS Deep Learning AMI標示為「Support 援」，則會取得安全性修補程式。

AWS發布新框架版本時會發布哪些圖像？

我們在新版本的 TensorFlow 和 PyTorch發布後不久發布新的 DLAMI。這包括主要版本，主要次要版本和框架 major-minor-patch 版本。我們也會在新版本的驅動程式和程式庫上市時更新映像檔。如需影像維護的更多資訊，請參閱 [我的框架版本的主動支持何時結束？](#)

哪些圖像獲得新的 SageMaker/AWS功能？

新功能通常會在最新版本的 DLAMIs 中發行，適用於 PyTorch 和 TensorFlow 如需AWS有關新功能 SageMaker 或功能的詳細資訊，請參閱特定影像的版本說明。如需可用 DLAMI 的清單，請參閱 DLAMI 的[版本說明](#)。如需影像維護的更多資訊，請參閱 [我的框架版本的主動支持何時結束？](#)

如何在支持的框架表中定義當前版本？

「[AWS Deep Learning AMI架構 Support 政策](#)」表中的目前版本是指在上提供的AWS最新架構版本 GitHub。每個最新版本都包含 DLAMI 中的驅動程式、程式庫和相關套件的更新。如需影像維護的資訊，請參閱 [我的框架版本的主動支持何時結束？](#)

如果我正在運行的版本不在支持框架表中怎麼辦？

如果您執行的版本不在 [AWS Deep Learning AMIFramework Support 原則表格](#)中，您可能沒有最新的驅動程式、程式庫和相關套件。如需更多 up-to-date 版本，建議您使用您選擇的最新 DLAMI 升級至其中一個可支援的架構。如需可用 DLAMI 的清單，請參閱 DLAMI 的[版本說明](#)。

DLAMI 是否支持以前版本的 TensorFlow

沒有 我們 Support 每個架構最新主要版本的最新修補程式版本，從[AWS Deep Learning AMI架構支援原則表格](#)中所述 365 天起發 GitHub 行。如需更多資訊，請參閱[如果我正在運行的版本不在支持框架表中怎麼辦？](#)

如何找到支持的框架版本的最新補丁圖像？

若要將 DLAMI 與最新的架構版本搭配使用，請擷取 DLAMI ID 並使用它來使用 EC2 主控台啟動 DLAMI。如需擷取 AWS Deep Learning AMI ID 的 AWS CLI 命令範例，請參閱[AWS Deep Learning AMI目錄](#)中的深度學習架構一節。AWSCLI AMI 識別碼查詢也包含在[單一架構 DLAMI](#) 發行說明中。您選擇的架構版本必須在「[AWS Deep Learning AMI架構支援政策](#)」表格中標示為「Support 援」。

新影像的發行頻率是多久？

提供更新的補丁版本是我們的首要任務。我們通常會在最早的機會中創建修補的圖像。我們監控新修補的框架版本（例如，TensorFlow 2.9 至 TensorFlow 2.9.1）和新的次要發行版本（例如 TensorFlow 2.9 到 TensorFlow 2.10），並在最早的機會提供它們。當使用新版 CUDA 發行的 TensorFlow 現有版本時，我們會針對該版本發行新的 DLAMI，並支援新的 TensorFlow CUDA 版本。

當我的工作負載執行時，我的執行個體是否會被修補？

沒有 DLAMI 的修補程式更新不是「就地」更新。

您必須開啟新的 EC2 執行個體、遷移工作負載和指令碼，然後關閉先前的執行個體。

當新的修補或更新的框架版本可用時會發生什麼情況？

定期查看映像檔的版本說明頁面。我們建議您在有新的修補或更新的框架可用時升級到它們。如需可用 DLAMI 的清單，請參閱 DLAMI 的[版本說明](#)。

依賴關係是否更新而不更改框架版本？

我們更新依賴關係而不更改框架版本。不過，如果相依性更新造成不相容性，我們會建立不同版本的映像檔。請務必查看 [DLAMI 的版本說明](#)，以取得更新的相依性資訊。

我的框架版本的主動支持何時結束？

DLAMI 圖像是不可變的。一旦它們被創建，它們不會改變。對框架版本的主動支持結束有四個主要原因：

- [架構版本 \(修補程式\) 升級](#)
- [AWS 安全性修補](#)
- [修補程式日期結束 \(過期\)](#)
- [相依性 end-of-support](#)

Note

由於版本修補程式升級和安全性修補程式的頻率，我們建議您經常查看 DLAMI 的版本說明頁面，並在進行變更時進行升級。

架構版本 (修補程式) 升級

如果您有以 TensorFlow 2.7.0 為基礎的 DLAMI 工作負載，且 TensorFlow 版本為 2.7.1 GitHub，則會 AWS 發行具有 2.7.1 的新 DLAMI。TensorFlow 2. TensorFlow 7.1 版的新影像發行後，先前的 2.7.0 影像將不再有效維護。含 TensorFlow 2.7.0 的 DLAMI 不會收到進一步的修補程式。然後，TensorFlow 2.7 的 DLAMI 發行說明頁面會更新為最新資訊。每個次要修補程式都沒有個別的版本說明頁面。

由於修補程式升級而建立的新 DLAMI 會使用新的 [AMI ID](#) 指定。

AWS 安全性修補

如果您有以 TensorFlow 2.7.0 的映像為基礎的工作負載並 AWS 製作了安全性修補程式，則 DLAMI 的新版本會針對 2.7.0 發行。TensorFlow 先前版本 TensorFlow 2.7.0 的影像不再維護。如需詳細資訊，請參閱 [當我的工作負載執行時，我的執行個體是否會被修補？](#) 如需尋找最新 DLAMI 的步驟，請參閱 [如何找到支持的框架版本的最新補丁圖像？](#)

由於修補程式升級而建立的新 DLAMI 會使用新的 [AMI ID](#) 指定。

修補程式日期結束 (過期)

DLAMI 在 GitHub 發布日期後 365 天結束補丁日期。

對於 [多框架 DLAMI](#)，當其中一個框架版本更新時，需要具有更新版本的新 DLAMI。舊框架版本的 DLAMI 不再積極維護。

⚠ Important

當有一個重大的框架更新時，我們做一個例外。例如。如果 TensorFlow 1.15 更新為 TensorFlow 2.0，則我們會繼續支援最新版本的 TensorFlow 1.15 版本，為期兩年，或原始架構維護團隊退出支援後六個月，以較早的日期為準。GitHub

相依性 end-of-support

如果您在使用 Python 3.6 的 TensorFlow 2.7.0 DLAMI 映像檔上執行工作負載，且該版本的 Python 已標記為 end-of-support，則將不再維護以 Python 3.6 為基礎的所有 DLAMI 映像檔。同樣，如果像 Ubuntu 16.04 這樣的作業系統版本被標記為 end-of-support，那麼依賴於 Ubuntu 16.04 的所有 DLAMI 映像將不再被積極維護。

具有不再積極維護的框架版本的圖像是否會被修補？

沒有 不再主動維護的映像將不會有新版本。

如何使用較舊的框架版本？

[要使用帶有較舊框架版本的 DLAMI，請檢索 DLAMI ID 並使用它使用 EC2 控制台啟動 DLAMI。](#)如需擷取 AMI ID 的 AWS CLI 命令，請參閱深度學習 [AMI 目錄中的 AWS 深度學習 架構](#) 一節。AWS CLI AMI 識別碼查詢也包含在 [單一架構 DLAMI](#) 發行說明中。

如何保持 up-to-date 框架及其版本的支持更改？

使 up-to-date 用架構 [Support 原則表格 \(DLAMI 發行說明\)](#)，繼續使用 [DLAMI AWS Deep Learning AMI 架構](#) 和版本。

我是否需要商業授權才能使用 Anaconda 軟體庫？

Anaconda 轉向某些使用者的商業授權模式。積極維護的 DLAMI 已從 Anaconda 頻道遷移到公開可用的開源版本康達 ([康達鍛造](#))。

啟動和設定 DLAMI

如果您的進度已到這裡，您應該對於想啟動哪些 AMI 已有一些好主意。如果沒有，請在[AWS Deep Learning AMI 目錄](#)中尋找 DLAMI 及其相關硬體、架構和 ID 擷取，或檢視[特拉米的發行公告](#)。

您也應該知道您要選擇哪種執行個體類型和區域。如果沒有，請瀏覽[選取 DLAMI 的例證類型](#)。

Note

我們將在實例中使用 p3.16xlarge 作為默認實例類型。您只需將其取代為您想使用的任何執行個體類型。

Important

如果您打算使用 Elastic Inference，您必須在啟動 DLAMI 之前完成 [Elastic Inference 設定](#)。

主題

- [步驟 1：啟動 DLAMI](#)
- [步驟 2：Connect 至 DLAMI](#)
- [步驟 3：測試您的 DLAMI](#)
- [步驟 4：管理您的 DLAMI 實例](#)
- [清除](#)
- [設定 Jupyter 筆記本伺服器](#)

步驟 1：啟動 DLAMI

Note

在本逐步解說中，我們可能會針對深度學習 AMI (Ubuntu 18.04) 進行參考 (Ubuntu 18.04)。即使您選擇了其他 DLAMI，也應該能夠遵循本指南。

1. [找到您的 DLAMI 的身份證](#)

2. [從 DLAMI 啟動 Amazon EC2 執行個體](#)

您將使用 Amazon EC2 主控台。按照中詳細說明進行操作[從 Amazon EC2 主控台啟動](#)

Tip

CLI 選項：如果您選擇使用 AWS CLI 啟動 DLAMI，則需要 AMI 的 ID、區域和執行個體類型以及安全性權杖資訊。請確認您已準備好 AMI 和執行個體 ID。如果尚未設定 AWS CLI，請先使用[安裝 AWS 命令列介面的指南進行設定](#)。

3. 完成其中一個選項的步驟後，請等待執行個體準備就緒。這通常只需要幾分鐘的時間。您可以在 [EC2 主控台](#) 中驗證執行個體的狀態。

擷取密碼 DLAMI 碼

每個 AMI 都擁有唯一識別碼 (ID)。您可以使用 AWS 命令列介面 (AWSCLI) 查詢您選擇的 DLAMI 的識別碼。如果您尚未 AWS CLI 安裝，請參閱[開始使用 AWS CLI](#)。

Note

提醒：您可以在[AWS Deep Learning AMI 目錄](#)中找到所有 DLAMI 及其相關處理器/加速器、作業系統、運算架構、推薦的 Amazon EC2 執行個體系列、支援狀態和 ID 擷取查詢。[特拉米的發行公告](#)如需其他資訊 (驅動程式、Python 版本、Amazon EBS 類型)，請參閱中的 DLAMI 版本說明。

1. 請確定已設定您的 AWS 認證。

```
aws configure
```

2. 使用以下命令檢索 DLAMI 的 ID 或查找 AWS Deep Learning AMI 目錄中提供的查詢。

```
aws ec2 describe-images --region us-east-1 --owners amazon \  
--filters 'Name=name,Values=Deep Learning AMI (Ubuntu 18.04) Version ??.' \  
'Name=state,Values=available' \  
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Note

您可以為特定架構指定發行版本，或以問號取代版本號碼以取得最新版本。

3. 輸出格式應類似以下內容：

```
ami-094c089c38ed069f2
```

複製此 DLAMI ID，然後按 **q** 退出提示。

後續步驟

[從 Amazon EC2 主控台啟動](#)

從 Amazon EC2 主控台啟動

Note

若要使用 Elastic Fabric Adapter (EFA) 啟動執行個體，請參閱[這些步驟](#)。

1. 開啟 [EC2 主控台](#)。
2. 請注意最上層導覽中您目前的區域。如果這不是您想要的 AWS 區域，請在繼續之前更改此選項。如需詳細資訊，請參閱 [EC2 區域](#)。
3. 選擇 Launch Instance (啟動執行個體)。
4. 輸入執行個體的名稱，然後選取適合您的 DLAMI。
 - a. 在「我的 AMI」中尋找現有的 DLAMI，或選擇「快速入門」。
 - b. 通過 DLAMI ID 搜索。瀏覽選項，然後選擇您的選擇。
5. 選擇執行個體類型。您可以在 AWS Deep Learning AMI 目錄中找到 DLAMI 的建議例證族群。如需 DLAMI 執行個體類型的一般建議，請參閱 [執行個體選擇](#)。

Note

如果要使用 [Elastic Inference \(EI\)](#)，請按一下設定執行個體詳細資訊，選取新增 Amazon EI 加速器，然後選取 Amazon EI 加速器的大小。

6. 選擇 Launch Instance (啟動執行個體)。

Tip

查看深度學習 [入門使用深度學習 AMI](#)，透過螢幕擷取畫面逐步解說。AWS

後續步驟

[步驟 2：Connect 至 DLAMI](#)

步驟 2：Connect 至 DLAMI

Connect 至您從用戶端 (視窗、MacOS 或 Linux) 啟動的 DLAMI。如需詳細資訊，請參閱 [《Linux 執行個體的 Amazon EC2 使用者指南》](#) 中的 [Connect 到 Linux 執行個體](#)。

如果您想在登入後設定 Jupyter，請將 SSH 登入命令的副本保持在方便取得之處。您將使用其變化版本來連接到 Jupyter 網頁。

後續步驟

[步驟 3：測試您的 DLAMI](#)

步驟 3：測試您的 DLAMI

根據您的 DLAMI 版本，您有不同的測試選項：

- [使用康達深度學習 AMI](#)— 去 [搭配康達使用深度學習 AMI](#)。
- [深度學習基礎 AMI](#)— 請參閱所需架構的安裝文件。

您也可以建立 Jupyter 筆記本、試用教學課程，或開始在 Python 中編輯程式碼。如需詳細資訊，請參閱 [設定 Jupyter 筆記本伺服器](#)。

步驟 4：管理您的 DLAMI 實例

當有可用的修補程式與更新時請務必立即套用，以保持您的作業系統和其他已安裝軟體處於最新狀態。

如果您使用的是亞馬遜 Linux 或 Ubuntu，當您登錄到 DLAMI 時，系統會通知您是否有更新，並查看更新說明。如需 Amazon Linux 維護的詳細資訊，請參閱[更新執行個體軟體](#)。對於 Ubuntu 執行個體，請參閱官方 [Ubuntu 文件](#)。

在 Windows 上，請定期檢查 Windows Update 是否有軟體和安全性更新。如果您願意，可以讓更新自動套用。

Important

有關崩潰和幽靈漏洞以及如何修補作業系統以防範這些漏洞的詳細資訊，請參閱[安全性公告 AWS -2018-013](#)。

清除

當您不再需要 DLAMI 時，您可以停止或終止它，以避免產生持續收費。停止執行個體仍會予以保留，因此您可以稍後恢復它。您的組態、檔案和其他非揮發性資訊會存放在 Amazon S3 上的磁碟區中。您需要支付小型 S3 費用以便在執行個體停止時保留磁碟區，但您無需支付當其處於已停止狀態時的運算資源費用。當您再次啟動執行個體，它會掛載該磁碟區，您的資料即可供使用。如果您終止執行個體，它就會消失，您再也無法將其啟動。您的資料實際上仍存放在 S3 中，因此，若要防止任何進一步的費用，您需要同時刪除磁碟區。[如需詳細說明，請參閱《Linux 執行個體的 Amazon EC2 使用者指南》](#)中的執行個體類型。

設定 Jupyter 筆記本伺服器

Jupyter 筆記本伺服器可讓您從 DLAMI 執行個體建立和執行 Jupyter 筆記本。使用 Jupyter 筆記本，您可以在使用 AWS 基礎結構和存取 DLAMI 中內建的套件時，進行機器學習 (ML) 實驗以進行訓練和推論。如需 Jupyter 筆記本的詳細資訊，請參閱 [Jupyter 筆記本文件](#)。

若要設定 Jupyter 筆記本伺服器，您必須：

- 在您的 Amazon EC2 DLAMI 執行個體上設定 Jupyter 筆記型電腦伺服器。
- 設定用戶端，讓您可以連接到 Jupyter 筆記本伺服器。我們提供適用於 Windows、macOS 和 Linux 用戶端的設定說明。
- 登入 Jupyter 筆記本伺服器，測試設定。

若要完成設定 Jupyter 的步驟，請遵循下列主題中的指示。設定 Jupyter 筆記本伺服器後，請參閱以[執行 Jupyter 筆記本教學課程](#)取得有關執行 DLAMI 中出貨的範例筆記本的資訊。

主題

- [保護您的 Jupyter 伺服器](#)
- [啟動 Jupyter 筆記本伺服器](#)
- [設定用戶端以連接到 Jupyter 伺服器](#)
- [登入 Jupyter 筆記本伺服器進行測試](#)

保護您的 Jupyter 伺服器

我們在此以 SSL 和自訂密碼設定 Jupyter。

Connect 到 Amazon EC2 執行個體，然後完成下列程序。

設定 Jupyter 伺服器

1. Jupyter 會提供密碼公用程式。執行以下命令，並在系統提示時輸入您慣用的密碼。

```
$ jupyter notebook password
```

輸出看起來像這樣：

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

2. 建立自簽 SSL 憑證。依照提示填寫您認為適合的地區。如果您希望將提示保留空白，您必須輸入 `.`。您的答案不會影響憑證的功能。

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

Note

您可能想要建立第三方所簽署的一般 SSL 憑證，這不會導致瀏覽器向您顯示安全警告。此程序涉及更多層面。如需詳細資訊，請瀏覽 [Jupyter 的文件](#)。

後續步驟

[啟動 Jupyter 筆記本伺服器](#)

啟動 Jupyter 筆記本伺服器

您現在可以登入執行個體並執行以下命令 (使用您在上一個步驟建立的 SSL 憑證)，藉此啟動 Jupyter 伺服器。

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

伺服器啟動後，您現在可以從您的用戶端電腦透過 SSH 通道連接到該伺服器。當伺服器執行時，您會看到 Jupyter 的一些輸出，確認伺服器正在執行中。此時，請忽略您可以透過 localhost URL 存取伺服器的標註，因為在您建立通道前，該動作並不可行。

Note

當您使用 Jupyter Web 界面切換架構時，Jupyter 會為您處理環境切換。您可以在 [使用 Jupyter 切換環境](#) 中找到更多資訊。

後續步驟

[設定用戶端以連接到 Jupyter 伺服器](#)

設定用戶端以連接到 Jupyter 伺服器

設定用戶端連接到 Jupyter 筆記本伺服器後，您可以在您的工作空間中建立和存取該伺服器上的筆記本，並在伺服器上執行您的深度學習程式碼。

如需設定資訊，請選擇以下其中一個連結。

主題

- [設定 Windows 用戶端](#)
- [設定 Linux 或 macOS 用戶端](#)

設定 Windows 用戶端

準備

請確認您具備設定 SSH 通道所需的以下資訊：

- Amazon EC2 執行個體的 Amazon EC2 執行個體的公有 DNS 名稱。您可以在 EC2 主控台中找到公有 DNS 名稱。
- 私有金鑰檔案的金鑰對。如需存取 key pair 的詳細資訊，請參閱適用於 Linux 執行個體的 [Amazon EC2 Linux](#) EC2 Linux EC2 執行個體的 Amazon EC2 Linux EC2 Linux EC2 執行個體。

從 Windows 用戶端使用 Jupyter 筆記本

請參閱這些從 Windows EC2 Windows Client Windows Amazon EC2 執行個體的 Windows EC2。

1. [疑難排解連線至執行個體](#)
2. [使用 PuTTY 從 Windows 連接至您的 Linux 執行個體](#)

若要建立執行中 Jupyter 伺服器的通道，建議方法是在 Windows 用戶端上安裝 Git Bash，然後依照 Linux/macOS 用戶端指示操作。不過，您可以使用您所要的方法，透過連接埠映射開啟 SSH 通道。如需詳細資訊，請參閱 [Jupyter 的文件](#)。

後續步驟

[設定 Linux 或 macOS 用戶端](#)

設定 Linux 或 macOS 用戶端

1. 開啟 終端機。
2. 執行下列命令，將本機連接埠 8888 上的所有請求轉送到遠端 Amazon EC2 執行個體上的連接埠 8888。取代金鑰的位置以存取 Amazon EC2 執行個體和 Amazon EC2 執行個體的公有 DNS 名稱，以更新命令。請注意，對於 Amazon Linux AMI，使用者名稱為 `ec2-user` 而非 `ubuntu`。

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


此命令，即可開啟正在執行 Jupyter 筆記本伺服器的客戶端和正在執行 Jupyter 筆記本伺服器的 Amazon EC2 執行個體。

後續步驟

[登入 Jupyter 筆記本伺服器進行測試](#)

登入 Jupyter 筆記本伺服器進行測試

現在您已準備好登入 Jupyter 筆記本伺服器。

您的下一個步驟是透過您的瀏覽器，測試伺服器的連線。

1. 在瀏覽器的位址列中輸入下列 URL，或按一下此連結：<https://localhost:8888>
2. 使用自簽 SSL 憑證，您的瀏覽器會警告您並提示您避免繼續瀏覽網站。



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](#)



Back to safety

因為您自行這麼設定，所以繼續操作安全無虞。視您的瀏覽器而定，您會收到「進階」、「顯示詳細資訊」或類似按鈕。



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 \(unsafe\)](#)

按一下此項，然後按一下「繼續 localhost」連結。如果連線成功，您會看到 Jupyter 筆記本伺服器網頁。此時，系統會要求您提供先前設定的密碼。

現在您可以存取在 DLAMI 上執行的 Jupyter 筆記型電腦伺服器。您可以建立新的筆記本或執行所提供的[教學課程](#)。

使用 DLAMI

主題

- [搭配康達使用深度學習 AMI](#)
- [使用深度學習基礎 AMI](#)
- [執行 Jupyter 筆記本教學課程](#)
- [教學課程](#)

下列各節說明如何使用具有 Conda 的深度學習 AMI 來切換環境、從每個架構執行範例程式碼，以及執行 Jupyter，以便您可以嘗試不同的筆記本教學課程。

搭配康達使用深度學習 AMI

主題

- [使用康達介紹深度學習 AMI](#)
- [登錄到您的 DLAMI](#)
- [啟動 TensorFlow 環境](#)
- [切換到 PyTorch Python 環境](#)
- [切換至 MXNet 3 環境](#)
- [移除環境](#)

使用康達介紹深度學習 AMI

Conda 是一套可在 Windows、macOS 和 Linux 上執行的開放原始碼套件管理系統以及環境管理系統。Conda 可快速安裝、執行和更新套件及其相依性。Conda 能輕鬆地建立、儲存、載入並在本機電腦的環境之間切換。

配備 Conda 的深度學習 AMI 已經設定，可讓您輕鬆在深度學習環境之間切換。以下說明引導您使用 conda 的一些基本命令。這些說明也協助您確認架構的基本匯入是否運作中，您是否可以使用架構執行幾個簡單操作。然後，您可以繼續閱讀 DLAMI 提供的更徹底的教程，或者在每個框架的項目站點上找到的框架示例。

登錄到您的 DLAMI

登入伺服器後，您會看到伺服器的「當日訊息」(MOTD)，描述您可用來切換不同深度學習架構的各種 Conda 命令。以下為範例 MOTD。您的特定 MOTD 可能會隨著 DLAMI 的新版本發布而有所不同。

Note

從 v28 版本開始，我們不再包括 CNTK，咖啡，Caffe2，西亞諾，查納和凱拉斯康達環境。AWS Deep Learning AMI 包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

```
=====
  _|  _|_ )
  _| (    /  Deep Learning AMI (Ubuntu 18.04) Version 40.0
  _|\___|___|
=====

Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1037-aws x86_64v)

Please use one of the following commands to start the required environment with the
framework of your choice:
for AWS MX 1.7 (+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN)
_____ source activate mxnet_p36
for AWS MX 1.8 (+Keras2) with Python3 (CUDA + and Intel MKL-DNN)
_____ source activate mxnet_latest_p37
for AWS MX(+AWS Neuron) with Python3
_____ source activate
aws_neuron_mxnet_p36
for AWS MX(+Amazon Elastic Inference) with Python3
_____ source activate amazonei_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA + and Intel MKL-DNN)
_____ source activate tensorflow_p37
for Tensorflow(+AWS Neuron) with Python3 _____
source activate aws_neuron_tensorflow_p36
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN)
_____ source activate tensorflow2_p36
for TensorFlow 2.3 with Python3.7 (CUDA + and Intel MKL-DNN) _____
source activate tensorflow2_latest_p37
for PyTorch 1.4 with Python3 (CUDA 10.1 and Intel MKL)
_____ source activate pytorch_p36
```

```
for PyTorch 1.7.1 with Python3.7 (CUDA 11.0 and Intel MKL)
_____ source activate pytorch_latest_p37
for PyTorch (+AWS Neuron) with Python3 _____
source activate aws_neuron_pytorch_p36
for base Python3 (CUDA 10.0)
_____ source
activate python3
```

每個 Conda 命令有下列模式：

```
source activate framework_python-version
```

例如，您可能會看到 for MXNet(+Keras1) with Python3 (CUDA 10.1)

_____ source activate mxnet_p36，這表示環境具有 MXNet、克拉斯 1、Python 3 和 CUDA 10.1。此外，您用來啟用此環境的命令為：

```
$ source activate mxnet_p36
```

啟動 TensorFlow 環境

Note

當您啟動第一個 Conda 環境時，請耐心等待它載入。搭配 Conda 的深度學習 AMI 會在架構第一次啟用時，自動為您的 EC2 執行個體安裝最佳化的架構版本。您應該不會遇到後續延遲。

1. 激活 Python 3 的 TensorFlow 虛擬環境。

```
$ source activate tensorflow_p37
```

2. 啟動 iPython 終端機。

```
(tensorflow_37)$ ipython
```

3. 運行一個快速的 TensorFlow 程序。

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

```
print(sess.run(hello))
```

您應該會看到「Hello, Tensorflow!」

接下來

[執行 Jupyter 筆記本教學課程](#)

切換到 PyTorch Python 環境

如果您仍在 IPython 控制台中，請使用`quit()`，然後準備切換環境。

- 激活 Python 3 的 PyTorch 虛擬環境。

```
$ source activate pytorch_p36
```

測試一些 PyTorch 代碼

若要測試您的安裝，請使用 Python 撰寫建立並列印陣列的 PyTorch 程式碼。

1. 啟動 iPython 終端機。

```
(pytorch_p36)$ ipython
```

2. 匯入 PyTorch。

```
import torch
```

您可能會看到有關第三方套件的警告訊息。您可以忽略。

3. 使用隨機初始化的元素創建一個 5x3 矩陣。列印陣列。

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

確認結果。

```
tensor([[0.3105, 0.5983, 0.5410],
        [0.0234, 0.0934, 0.0371],
```

```
[0.9740, 0.1439, 0.3107],  
[0.6461, 0.9035, 0.5715],  
[0.4401, 0.7990, 0.8913]])
```

切換至 MXNet 3 環境

如果您仍在 IPython 控制台中，請使用 `quit()`，然後準備切換環境。

- 啟用適用於 Python 3 的 MXNet 虛擬環境。

```
$ source activate mxnet_p36
```

測試一些 MXNet 程式碼

為了測試您的安裝，請使用 Python 撰寫 MXNet 程式碼，以使用 NDAarray API 來建立和列印陣列。如需詳細資訊，請參閱 [NDAarray API](#)。

1. 啟動 iPython 終端機。

```
(mxnet_p36)$ ipython
```

2. 匯入 MXNet。

```
import mxnet as mx
```

您可能會看到有關第三方套件的警告訊息。您可以忽略。

3. 建立 5x5 矩陣、NDAarray 的執行個體，元素初始化為 0。列印陣列。

```
mx.ndarray.zeros((5,5)).asnumpy()
```

確認結果。

```
array([[ 0.,  0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.,  0.]], dtype=float32)
```

您可以在 MXNet 教學課程部分找到更多 MXNet 範例。

移除環境

如果 DLAMI 上的空間不足，您可以選擇解除安裝不使用的 Conda 套件：

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

使用深度學習基礎 AMI

使用深度學習基礎 AMI

基礎 AMI 隨附一個 GPU 驅動程式的基礎平台和加速程式庫，可用以部署您自己的自訂深度學習環境。默認情況下，AMI 配置了 NVIDIA CUDA 11.0 環境。您也可以在不同版本的 CUDA 之間切換。請參閱下列關於如何操作的說明。

設定 CUDA 版本

您可以通過運行 NVIDIA 的 `nvcc` 程序來驗證 CUDA 版本。

```
nvcc --version
```

您可以使用以下 `bash` 命令選擇並驗證特定的 CUDA 版本：

```
sudo rm /usr/local/cuda
sudo ln -s /usr/local/cuda-11.0 /usr/local/cuda
```

如需詳細資訊，請參閱[基本 DLAMI 版本說明](#)。

執行 Jupyter 筆記本教學課程

教程和示例隨每個深度學習項目的來源一起提供，在大多數情況下，它們將在任何 DLAMI 上運行。如果您選擇[使用康達深度學習 AMI](#)，您可以獲得數個已設定好可供試用的精選教學課程的額外好處。

⚠ Important

若要執行 DLAMI 上安裝的 Jupyter 筆記本教學課程，您必須執行。[設定 Jupyter 筆記本伺服器](#)

Jupyter 伺服器開始執行後，您可以透過 Web 瀏覽器執行教學課程。如果您使用 Conda 執行深度學習 AMI，或者您已經設定了 Python 環境，您可以從 Jupyter 筆記本介面切換 Python 核心。選擇適當的核心，再嘗試執行架構特定的教學課程。為使用 Conda 的深度學習 AMI 的使用者提供了進一步的範例。

ℹ Note

許多教程都需要額外的 Python 模塊，這些模塊可能不會在 DLAMI 上設置。如果出現類似的錯誤 "xyz module not found"，請登入 DLAMI，如上所述啟動環境，然後安裝必要的模組。

ℹ Tip

深度學習教學課程和範例通常依賴一或多個 GPU。如果您的執行個體類型沒有 GPU，您可能需要變更一些範例中的程式碼，才能讓其執行。

瀏覽安裝教學課程

一旦您登錄到 Jupyter 伺服器並可以看到教程目錄（僅在具有 Conda 的深度學習 AMI 上），您將看到每個框架名稱的教程文件夾。如果您沒有看到列出的框架，那麼當前 DLAMI 上該框架的教程不可用。按一下架構的名稱以查看列出的教學課程，然後按一下教學課程來啟動它。

第一次使用 Conda 在深度學習 AMI 上執行筆記型電腦時，它會想要知道您想要使用哪個環境。它會提示您從清單中選取。每個環境都根據此模式命名：

Environment (conda_framework_python-version)

例如，您可能會看到 Environment (conda_mxnet_p36)，這表示該環境有 MXNet 和 Python 3。此項目的其他變化包括 Environment (conda_mxnet_p27)，表示環境具有 MXNet 和 Python 2。

ℹ Tip

如果您擔心哪個版本的 CUDA 處於活動狀態，第一次登錄 DLAMI 時可以在 MOTD 中查看它。

使用 Jupyter 切換環境

如果您決定試用不同架構的教學課程，請務必確認目前執行的核心。此資訊可以在 Jupyter 界面的右上方查看，位於登出按鈕下方。您可以按一下 Jupyter 功能表項目 Kernel (核心)、按一下 Change Kernel (變更核心)，然後按一下最適合您正在執行之筆記本的環境，即可變更任何開啟筆記本的核心。

此時，您必須重新執行任何儲存格，因為核心中的變更將會清除您先前執行之任何項目的狀態。

Tip

在架構之間切換或許很有趣也富有教育意義，但您可能會用盡記憶體。如果開始出現錯誤，請查看 Jupyter 伺服器正在執行的終端機視窗。這裡有有用的消息和錯誤記錄，您可能會看到錯 out-of-memory 誤。若要修正此問題，您可以移至 Jupyter 伺服器首頁，按一下 Running (執行中) 標籤，然後按一下每個可能仍在背景執行並耗盡所有記憶體之教學課程的 Shutdown (關閉)。

接下來

如需每個架構的更多範例和範本程式碼，請按一下 Next (下一步) 或繼續前往 [阿帕奇 MXnet \(孵化中 \)](#)。

教學課程

以下是如何搭配 Conda 軟體使用深度學習 AMI 的教學課程。

主題

- [10 分鐘教學課程](#)
- [啟用架構](#)
- [偵錯和視覺化](#)
- [分散式訓練](#)
- [Elastic Fabric Adapter](#)
- [GPU 監控和最佳化](#)
- [具有 DL AWS AMI 的推論芯片](#)
- [重力子 DLAMI](#)
- [哈瓦那 DLAMI](#)
- [Inference](#)

- [使用架構與 ONNX](#)
- [模型服務](#)

10 分鐘教學課程

- [啟動一個 AWS Deep Learning AMI \(在 10 分鐘內\)](#)
- [在 Amazon EC2 上使用 DLC 訓練深度學習模型 \(只需 10 分鐘\)](#)

啟用架構

以下是使用 Conda 在深度學習 AMI 上安裝的深度學習架構。按一下某個架構，以了解如何啟用。

主題

- [阿帕奇 MXnet \(孵化中\)](#)
- [Caffe2](#)
- [Chainer](#)
- [CNTK](#)
- [Keras](#)
- [PyTorch](#)
- [TensorFlow](#)
- [TensorFlow 2](#)
- [TensorFlow 與霍洛沃德](#)
- [TensorFlow 2 與霍羅沃德](#)
- [Theano](#)

阿帕奇 MXnet (孵化中)

啟動阿帕奇 MXnet (孵化中)

本教學課程說明如何在執行深度學習 AMI (Conda 上的 DLAMI) 執行深度學習 AMI 的執行個體上啟動 MXNet，以及如何執行 MXNet 程式。

當框架的穩定 Conda 軟件包發布時，它已在 DLAMI 上進行測試並預先安裝。如果您想要執行最新、未經測試的每夜組建，您可以手動[安裝 MXNet 的每夜組建 \(實驗性\)](#)。

使用康達在 DLAMI 上執行 MXNet

1. 要激活框架，請使用 Conda 打開 DLAMI 的 Amazon 彈性計算雲 (亞馬遜 EC2) 實例。
 - 對於 Python 3 (含 CUDA 9.0 和 MKL-DNN) 上的 MXNet 與 Keras 2，執行此命令：

```
$ source activate mxnet_p36
```

- 對於 Python 2 (含 CUDA 9.0 和 MKL-DNN) 上的 MXNet 與 Keras 2，執行此命令：

```
$ source activate mxnet_p27
```

2. 啟動 iPython 終端機。

```
(mxnet_p36)$ ipython
```

3. 執行快速 MXNet 程式。建立 5x5 矩陣、NDArray 的執行個體，元素初始化為 0。列印陣列。

```
import mxnet as mx
mx.ndarray.zeros((5,5)).asnumpy()
```

4. 確認結果。

```
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]], dtype=float32)
```

安裝 MXNet 的每夜組建 (實驗性)

您可以使用 Conda 在深度學習 AMI 上將最新的 MXNet 組建安裝到 MXNet Conda 環境中的一個或兩個環境中。

從每夜組建安裝 MXNet

1. • 若為 Python 3 MXNet 環境，執行此命令：

```
$ source activate mxnet_p36
```

- 若為 Python 2 MXNet 環境，執行此命令：

```
$ source activate mxnet_p27
```

2. 移除目前安裝的 MXNet。

Note

其餘步驟假設您使用的是 mxnet_p36 環境。

```
(mxnet_p36)$ pip uninstall mxnet-cu90mkl
```

3. 安裝 MXNet 的最新每夜組建。

```
(mxnet_p36)$ pip install --pre mxnet-cu90mkl
```

4. 為了確認您是否成功安裝最新的每夜組建，請啟動 IPython 終端機並檢查 MXNet 版本。

```
(mxnet_p36)$ ipython
```

```
import mxnet
print (mxnet.__version__)
```

輸出中將印出最新 MXNet 穩定版本。

其他教學

您可以在具有 Conda 教程的深度學習 AMI 文件夾中找到更多教程，該文件夾位於 DLAMI 的主目錄中。

1. [使用阿帕奇 MXNet \(孵化 \) 進行 50 模型的推論 ResNet](#)
2. [使用阿帕奇 MXNet \(孵化 \) 進行 ONNX 模型的推論](#)
3. [Apache MXNet 專用模型伺服器 \(MMS\)](#)

有關更多教程和示例，請參閱框架的官方 Python 文檔，[用於 MXNet 的 Python API](#) 或 [阿帕奇 MXNet 網站](#)。

Caffe2

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

Caffe2 教學課程

若要啟用架構，請使用 Conda 遵循深度學習 AMI 上的這些指示。

只有 Python 2 與 CUDA 9 與 cuDNN 7 選項：

```
$ source activate caffe2_p27
```

啟動 iPython 終端機。

```
(caffe2_p27)$ ipython
```

執行快速 Caffe2 程式。

```
from caffe2.python import workspace, model_helper
import numpy as np
# Create random tensor of three dimensions
x = np.random.rand(4, 3, 2)
print(x)
print(x.shape)
workspace.FeedBlob("my_x", x)
x2 = workspace.FetchBlob("my_x")
print(x2)
```

您應該會看到列印初始 numpy 隨機陣列，然後將這些內容載入到 Caffe2 Blob。請注意，載入後它們是相同的。

其他教學

有關更多教程和示例，請參閱框架的官方 Python 文檔，[用於 Caffe2 的 Python API](#) 和 [Caffe2 網站](#)。

Chainer

Note

從 v28 版本 AWS Deep Learning AMI 開始，我們不再包含 Chainer Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

[Chainer](#) 是一種靈活的 Python 型架構，讓您輕鬆、直覺地編寫複雜的神經網路架構。Chainer 可讓您輕鬆地使用多 GPU 執行個體來進行訓練。Chainer 也會自動記錄結果、將損失和準確性製成圖表，並產生輸出以供使用[運算圖](#)來視覺化呈現神經網路。它包含在孔達 (DLAMI AMI 與康達) 的深度學習 AMI 中。

啟用 Chainer

1. 透過 Conda Connect 至執行深度學習 AMI 的執行個體。有關如何選擇[the section called “執行個體選擇”](#)或連接到執行個體，請參閱或 [Amazon EC2 文件](#)。
2. • 啟用 Python 3 Chainer 環境：

```
$ source activate chainer_p36
```

- 啟用 Python 2 Chainer 環境：

```
$ source activate chainer_p27
```

3. 啟動 iPython 終端機：

```
(chainer_p36)$ ipython
```

4. 測試匯入 Chainer，確定是否正常運作：

```
import chainer
```

您可能會看到一些警告訊息，但沒有錯誤。

詳細資訊

- 嘗試看看 [Chainer](#) 的教學課程。
- 您稍早下載之來源中的 Chainer 範例資料夾包含多個範例。試試看這些範例，了解如何執行。
- 若要進一步了解 Chainer，請參閱 [Chainer 文件網站](#)。

CNTK

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

啟動 CNTK

本教學課程說明如何在執行深度學習 AMI (Conda 上的 DLAMI) 的執行個體上啟動 CNTK，以及如何執行 CNTK 程式。

當框架的穩定 Conda 軟件包發布時，它已在 DLAMI 上進行測試並預先安裝。如果您想要執行最新、未經測試的每夜組建，您可以手動 [安裝 CNTK 每夜構建 \(實驗性\)](#)。

與康達一起在 DLAMI 機上執行 CNTK

1. 若要啟用 CNTK，請使用 Conda 開啟 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。
 - 對於 Python 3 與 CUDA 9 與 cuDNN 7：

```
$ source activate cntk_p36
```

- 對於 Python 2 與 CUDA 9 與 cuDNN 7：

```
$ source activate cntk_p27
```

2. 啟動 iPython 終端機。

```
(cntk_p36)$ ipython
```

3.
 - 如果您具有 CPU 執行個體，請執行此快速 CNTK 程式。


```
import cntk as C
C.__version__
c = C.constant(3, shape=(2,3))
c.asarray()
```

您應該會看到 CNTK 版本，接著是 3 的 2x3 陣列輸出。

- 如果您有 GPU 執行個體，您可以使用以下程式碼範例來測試它。如果 CNTK 可以存取 GPU，您可預期收到 True 的結果。

```
from cntk.device import try_set_default_device, gpu
try_set_default_device(gpu(0))
```

安裝 CNTK 每夜構建 (實驗性)

您可以使用 Conda 在深度學習 AMI 上將最新的 CNTK 內建安裝到 CNTK Conda 環境中的一個或兩個環境中。

從每夜組建安裝 CNTK

1. • 對於 Python 3 (含 CUDA 9.0 及 MKL-DNN) 上的 CNTK 及 Keras 2，執行此命令：

```
$ source activate cntk_p36
```

- 對於 Python 2 (含 CUDA 9.0 及 MKL-DNN) 上的 CNTK 及 Keras 2，執行此命令：

```
$ source activate cntk_p27
```

2. 其餘步驟假設您使用的是 cntk_p36 環境。移除目前安裝的 CNTK。

```
(cntk_p36)$ pip uninstall cntk
```

3. 若要安裝 CNTK 每夜組建，您首先需要從 [CNTK 每夜網站](#) 找到您要安裝的版本。
4. • (GPU 執行個體的選項) - 若要安裝每夜組建，您應使用下列命令，取代成您需要的組建：

```
(cntk_p36)$ pip install https://cntk.ai/PythonWheel/GPU/latest-nightly-build
```

將前一個命令中的 URL 取代為目前 Python 環境的 GPU 版本。

- (CPU 執行個體的選項) - 若要安裝每夜組建，您應使用下列命令，取代成您需要的組建：

```
(cntk_p36)$ pip install https://cntk.ai/PythonWheel/CPU-Only/latest-nightly-build
```

將前一個命令中的 URL 取代為目前 Python 環境的 CPU 版本。

5. 若要確認您是否成功安裝最新的每夜組建，請啟動 IPython 終端機並檢查 CNTK 版本。

```
(cntk_p36)$ ipython
```

```
import cntk
print (cntk.__version__)
```

輸出應該會列印類似於 2.6-rc0.dev20181015 的內容

其他教學

[如需更多教學課程和範例，請參閱架構的官方 Python 文件、CNTK 的 Python API 或 CNTK 網站。](#)

Keras

Keras 教學課程

1. 若要啟用架構，請在 [the section called “康达·DLAMI”](#) CLI 中使用這些命令。

- 對於 Python 3 與 CUDA 9 與 cuDNN 7 上的 Keras 2 與 MXNet 後端：

```
$ source activate mxnet_p36
```

- 對於 Python 2 與 CUDA 9 與 cuDNN 7 上的 Keras 2 與 MXNet 後端：

```
$ source activate mxnet_p27
```

- 對於切拉斯 2 與 Python 3 與 CUDA 9 與 CUDNN 7 的 TensorFlow 後端：

```
$ source activate tensorflow_p36
```

- 對於切拉斯 2 在 Python 2 與 CUDA 9 與 cuDNN N 7 的 TensorFlow 後端：

```
$ source activate tensorflow_p27
```

2. 若要測試匯入 Keras 以驗證哪些後端是啟用的，請使用下列命令：

```
$ ipython
import keras as k
```

畫面上應會出現下列內容：

```
Using MXNet backend
```

如果使用 Keras TensorFlow，則會顯示以下內容：

```
Using TensorFlow backend
```

Note

如果收到錯誤，或是仍在使用錯誤的後端，您可以手動更新 Keras 組態。編輯 `~/.keras/keras.json` 檔案並將後端設定變更為 `mxnet` 或 `tensorflow`。

其他教學

- 對於使用 Keras 與 MXNet 後端的多重 GPU 教學課程，請試試看 [Keras-MXNet 多重 GPU 訓練教學課程](#)。
- 您可以在具有 Conda `~/examples/keras-mxnet` 的深度学习 AMI 目錄中找到具有 MXNet 後端的 Keras 範例。
- 您可以在具有 Conda `~/examples/keras` 目錄的深度学习 AMI 中找到具有 TensorFlow 後端功能的 Keras 範例。
- 如需其他教學課程和範例，請參閱 [Keras](#) 網站。

PyTorch

啟動 PyTorch

當框架的穩定 Conda 軟件包發布時，它已在 DLAMI 上進行測試並預先安裝。如果您想要執行最新、未經測試的每夜組建，您可以手動 [安裝 PyTorch 的夜間構建 \(實驗性\)](#)。

若要啟用目前安裝的架構，請依照 Conda 的深度學習 AMI 上的這些指示進行。

對 PyTorch 於使用 CUDA 10 和 MKL-DNN 的 Python 3，運行以下命令：

```
$ source activate pytorch_p36
```

對 PyTorch 於使用 CUDA 10 和 MKL-DNN 的 Python 2，運行以下命令：

```
$ source activate pytorch_p27
```

啟動 iPython 終端機。

```
(pytorch_p36)$ ipython
```

運行一個快速的 PyTorch 程序。

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

您應該會看到列印初始隨機陣列、接著是其大小，然後新增另一個隨機陣列。

安裝 PyTorch 的夜間構建 (實驗性)

如何 PyTorch 從夜間構建安裝

您可以使用 Conda 將最新 PyTorch 版本安裝到深度學習 AMI 上的一個或兩個 PyTorch Conda 環境中。

1. • (Python 3 的選項) - 激活 Python 3 PyTorch 環境：

```
$ source activate pytorch_p36
```

• (Python 2 的選項) - 激活 Python 2 PyTorch 環境：

```
$ source activate pytorch_p27
```

- 其餘步驟假設您使用的是 `pytorch_p36` 環境。刪除當前安裝的 PyTorch :

```
(pytorch_p36)$ pip uninstall torch
```

- (GPU 實例的選項) - PyTorch 使用 CUDA 10.0 安裝最新的夜間版本 :

```
(pytorch_p36)$ pip install torch_nightly -f https://download.pytorch.org/whl/nightly/cu100/torch_nightly.html
```

- (CPU 執行個體的選項)-針對沒有 GPU 的執行個體安裝最新 PyTorch 的夜間組建 :

```
(pytorch_p36)$ pip install torch_nightly -f https://download.pytorch.org/whl/nightly/cpu/torch_nightly.html
```

- 要驗證您是否成功安裝了最新的夜間構建，請啟動 IPython 終端並檢查 PyTorch

```
(pytorch_p36)$ ipython
```

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

輸出應該會列印類似於 `1.0.0.dev20180922` 的內容

- 要驗證 PyTorch 每晚構建 PyTorch 是否與 MNIST 示例一起運行良好，您可以從的示例存儲庫中運行測試腳本 :

```
(pytorch_p36)$ cd ~
(pytorch_p36)$ git clone https://github.com/pytorch/examples.git pytorch_examples
(pytorch_p36)$ cd pytorch_examples/mnist
(pytorch_p36)$ python main.py || exit 1
```

其他教學

您可以在 DLAMI AMI 的主目錄中使用 Conda 教程文件夾中的深度學習 AMI 中找到更多教程。有關更多教程和示例，請參閱框架的官方 [PyTorch 文檔](#)，[文檔](#)和 [PyTorch](#) 網站。

- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

TensorFlow

啟動 TensorFlow

本教學課程說明如何 TensorFlow 在執行深度學習 AMI 的執行個體上使用 Conda (Conda 上的 DLAMI) 啟動並執行程式。TensorFlow

當框架的穩定 Conda 軟件包發布時，它已在 DLAMI 上進行測試並預先安裝。如果您想要執行最新、未經測試的每夜組建，您可以手動[安裝 TensorFlow 的夜間構建 \(實驗性\)](#)。

與康達一起 TensorFlow 在 DLAMI 上運行

1. 若要啟用 TensorFlow，請使用 Conda 開啟 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。

- 對於 TensorFlow 和奇拉斯 2 在 Python 3 與 CUDA 9.0 和 MKL-DNN，運行以下命令：

```
$ source activate tensorflow_p36
```

- 對於 TensorFlow 和奇拉斯 2 與 CUDA 9.0 和 MKL-DNN 的 Python 2，運行以下命令：

```
$ source activate tensorflow_p27
```

2. 啟動 iPython 終端機：

```
(tensorflow_p36)$ ipython
```

3. 運行一個 TensorFlow 程序來驗證它是否正常工作：

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

Hello, TensorFlow! 應該會出現在您的畫面上。

安裝 TensorFlow 的夜間構建 (實驗性)

您可以使用 Conda 將最新 TensorFlow 版本安裝到深度學習 AMI 上的一個或兩個 TensorFlow Conda 環境中。

TensorFlow 從夜間構建安裝

- 對於 Python 3 TensorFlow 環境，請運行以下命令：

```
$ source activate tensorflow_p36
```

- 對於 Python 2 TensorFlow 環境，請運行以下命令：

```
$ source activate tensorflow_p27
```

2. 刪除當前安裝的 TensorFlow。

Note

其餘步驟假設您使用的是 tensorflow_p36 環境。

```
(tensorflow_p36)$ pip uninstall tensorflow
```

3. 安裝的最新夜間組建。TensorFlow

```
(tensorflow_p36)$ pip install tf-nightly
```

4. 要驗證您是否成功安裝了最新的夜間構建，請啟動 IPython 終端並檢查 TensorFlow

```
(tensorflow_p36)$ ipython
```

```
import tensorflow
print (tensorflow.__version__)
```

輸出應該會列印類似於 1.12.0-dev20181012 的內容

其他教學

[TensorFlow 與霍洛沃德](#)

[TensorBoard](#)

[TensorFlow 服務](#)

如需教學課程，請參閱 DLAMI 主目錄 [Deep Learning AMI with Conda tutorials](#) 中名為的資料夾。

有關更多教程和示例，請參閱 [TensorFlow Python API](#) 的 TensorFlow 文檔或訪問 [TensorFlow](#) 網站。

TensorFlow 2

本教學課程說明如何在執行深度學習 AMI (Conda 上的 DLAMI) 執行深度學習 AMI 的執行個體上啟動 TensorFlow 2，並執行 2 程式。 TensorFlow

當框架的穩定 Conda 軟件包發布時，它已在 DLAMI 上進行測試並預先安裝。如果您想要執行最新、未經測試的每夜組建，您可以手動 [安裝 TensorFlow 2 的夜間構建 \(實驗性\)](#)。

啟動中 TensorFlow 2

與康達一起 TensorFlow 在 DLAMI 上運行

1. 若要啟用 TensorFlow 2，請使用 Conda 開啟 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。
2. 對於 TensorFlow 使用 CUDA 10.1 和 MKL-DNN 的 Python 3 上的 2 和克拉斯 2，請運行以下命令：

```
$ source activate tensorflow2_p36
```

3. 啟動 iPython 終端機：

```
(tensorflow2_p36)$ ipython
```

4. 運行一個 TensorFlow 2 程序來驗證它是否正常工作：

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

Hello, TensorFlow! 應該會出現在您的畫面上。

安裝 TensorFlow 2 的夜間構建 (實驗性)

您可以使用 Conda 將最新的 TensorFlow 2 版本安裝到深度學習 AMI 上的任何一個或兩個 Conda 環境中。TensorFlow

TensorFlow 從夜間構建安裝

1. 對於 Python 3 TensorFlow 2 環境，請運行以下命令：

```
$ source activate tensorflow2_p36
```

2. 刪除當前安裝的 TensorFlow。

Note

其餘步驟假設您使用的是 tensorflow2_p36 環境。

```
(tensorflow2_p36)$ pip uninstall tensorflow
```

3. 安裝的最新夜間組建。TensorFlow

```
(tensorflow2_p36)$ pip install tf-nightly
```

4. 要驗證您是否成功安裝了最新的夜間構建，請啟動 IPython 終端並檢查 TensorFlow

```
(tensorflow2_p36)$ ipython
```

```
import tensorflow
print (tensorflow.__version__)
```

輸出應該會列印類似於 2.1.0-dev20191122 的內容

其他教學

如需教學課程，請參閱 DLAMI 主目錄 Deep Learning AMI with Conda tutorials 中名為的資料夾。

有關更多教程和示例，請參閱 [TensorFlow Python API](#) 的 TensorFlow 文檔或訪問 [TensorFlow](#) 網站。

TensorFlow 與霍羅沃德

本教程介紹了如何 TensorFlow 使用康達在 AWS Deep Learning AMI (DLAMI) 上使用霍羅沃德激活。霍羅維德已預先安裝在康達環境中的. TensorFlow 建議使用 Python3 環境。

Note

只支援 P3.*、P2.* 和 G3.* 執行個體類型。

使用康達在 DLAMI 上激活 TensorFlow 和測試霍羅沃德

1. 使用 Conda 開啟 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。如需開始使用 DLAMI 的說明，請參閱 [the section called “如何開始使用 DLAMI”](#)
2. (推薦) 對於使用 CUDA 11 的 Python 3 上使用霍羅維德的 TensorFlow 1.15，請運行以下命令：

```
$ source activate tensorflow_p37
```

3. 啟動 iPython 終端機：

```
(tensorflow_p37)$ ipython
```

4. TensorFlow 使用 Horovod 測試匯入，以確認它是否正常運作：

```
import horovod.tensorflow as hvd
hvd.init()
```

您的畫面上可能會出現下列內容 (您可以忽略任何警告訊息)。

```
-----
[[55425,1],0]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:
```

```
Module: OpenFabrics (openib)
Host: ip-172-31-72-4
```

```
Another transport will be used instead, although this may result in
```

```
lower performance.  
-----
```

詳細資訊

- [TensorFlow 與霍羅沃德](#)
- 如需教學課程，請參閱 DLAMI 主目錄中的 `examples/horovod` 資料夾。
- 如需更多教學課程和範例，請參閱 [Horovod GitHub](#) 專案。

TensorFlow 2 與霍羅沃德

本教程介紹了如何使用康達在 AWS Deep Learning AMI (DLAMI) 上使用霍羅沃德激活 TensorFlow 2。霍羅維德已預先安裝在 2 的康達環境中。TensorFlow 建議使用 Python3 環境。

Note

只支援 P3.*、P2.* 和 G3.* 執行個體類型。

使用康達在 DLAMI 上激活 TensorFlow 2 並測試霍羅沃德

1. 使用 Conda 開啟 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。如需開始使用 DLAMI 的說明，請參閱 [the section called “如何開始使用 DLAMI”](#)
 - (推薦) 對於在具有 CUDA 10 的 Python 3 上使用霍羅維德的 TensorFlow 2，請運行以下命令：

```
$ source activate tensorflow2_p36
```

- 對於 TensorFlow 在具有 CUDA 10 的 Python 2 上使用霍羅維德的 2，請運行以下命令：

```
$ source activate tensorflow2_p27
```

2. 啟動 iPython 終端機：

```
(tensorflow2_p36)$ ipython
```

3. 使用 Horovod 測試匯入 TensorFlow 2，以確認它是否正常運作：

```
import horovod.tensorflow as hvd
hvd.init()
```

如果沒有收到任何輸出，那麼 Horovod 正常工作。您的畫面上可能會出現下列內容 (您可以忽略任何警告訊息)。

```
-----
[[55425,1],0]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: ip-172-31-72-4

Another transport will be used instead, although this may result in
lower performance.
-----
```

詳細資訊

- 如需教學課程，請參閱 DLAMI 主目錄中的 `examples/horovod` 資料夾。
- 如需更多教學課程和範例，請參閱 [Horovod GitHub](#) 專案。

Theano

Theano 教學課程

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

若要啟用架構，請使用 Conda 遵循深度學習 AMI 上的這些指示。

對於 Python 3 與 CUDA 9 與 cuDNN 7 中的 Theano + Keras :

```
$ source activate theano_p36
```

對於 Python 2 與 CUDA 9 與 cuDNN 7 中的 Theano + Keras :

```
$ source activate theano_p27
```

啟動 iPython 終端機。

```
(theano_p36)$ ipython
```

執行快速 Theano 程式。

```
import numpy
import theano
import theano.tensor as T
from theano import pp
x = T.dscalar('x')
y = x ** 2
gy = T.grad(y, x)
pp(gy)
```

您應該會看到 Theano 運算符號梯度。

其他教學

有關更多教程和示例，請參閱框架的官方文檔，[Theano Python API](#) 和 [Theano](#) 網站。

偵錯和視覺化

瞭解 DLAMI 的除錯和視覺化選項。按一下其中一個選項，了解如何使用它。

主題

- [MXBoard](#)
- [TensorBoard](#)

MXBoard

[MXBoard](#) 可讓您使用軟體以視覺方式檢查和解譯 MXNet 執行階段和圖形。TensorBoard 它會執行 Web 伺服器，提供網頁用以檢視 MXBoard 視覺化效果並與其互動。

MXNet TensorBoard、和 MXBoard 已預先安裝深度學習 AMI 與康達 (DLAMI 與康達)。在本教學課程中，您會使用 MXBoard 函數來產生相容的 TensorBoard 記錄。

主題

- [使用 MXNet 搭配 MXBoard](#)
- [詳細資訊](#)

使用 MXNet 搭配 MXBoard

產生相容的 MXBoard 記錄資料 TensorBoard

1. 使用 Conda Connect 到 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。
2. 啟用 Python 3 MXNet 環境。

```
$ source activate mxnet_p36
```

3. 準備 Python 指令碼，用於將正規運算子產生的資料寫入至事件檔案。資料會以降低標準差產生十次，每次都會寫入至事件檔案。您會看到資料分佈逐漸趨向於集中在平均值附近。請注意，您需指定日誌資料夾中的事件檔案。您將此文件夾路徑傳遞給 TensorBoard 二進製文件。

```
$ vi mxboard_normal.py
```

4. 在檔案中貼上以下內容，然後儲存檔案：

```
import mxnet as mx
from mxboard import SummaryWriter

with SummaryWriter(logdir='./logs') as sw:
    for i in range(10):
        # create a normal distribution with fixed mean and decreasing std
        data = mx.nd.normal(loc=0, scale=10.0/(i+1), shape=(10, 3, 8, 8))
        sw.add_histogram(tag='norml_dist', values=data, bins=200, global_step=i)
```

5. 執行指令碼。這將在 logs 資料夾中產生日誌，您可以用於視覺化。

```
$ python mxboard_normal.py
```

6. 現在，您必須切換到要使用的 TensorFlow 環境 TensorBoard 和 MXBoard 以視覺化記錄。這是 MXBoard 和 TensorBoard. 的必要相依性。

```
$ source activate tensorflow_p36
```

7. 傳遞日誌的位置給 tensorboard：

```
$ tensorboard --logdir=./logs --host=127.0.0.1 --port=8888
```

TensorBoard 在連接埠 8888 上啟動視覺化網頁伺服器。

8. 若想從本機瀏覽器輕鬆存取，您可以將 Web 伺服器連接埠變更為連接埠 80 或其他連接埠。無論您使用哪個連接埠，都必須在 EC2 安全群組中為 DLAMI 開啟此連接埠。您也可以使用連接埠轉送。如需變更安全群組設定和連接埠轉送的詳細資訊，請參閱[設定 Jupyter 筆記本伺服器](#)。預設設定將於後續步驟中說明。

Note

如果您需要同時執行 Jupyter 伺服器和 MXBoard 伺服器，請為每個伺服器使用不同的連接埠。

9. 在 EC2 執行個體上開啟連接埠 8888 (或您指派給視覺化 Web 伺服器的連接埠)。
 - a. [在 Amazon EC2 控制台中打開您的 EC2 實例，網址為 https://console.aws.amazon.com/ec2/。](https://console.aws.amazon.com/ec2/)
 - b. 在 Amazon EC2 主控台中，選擇「網路和安全性」，然後選擇「安全群組」。
 - c. 對於 Security Group (安全群組)，選擇最近建立的項目 (請參考描述中的時間戳記)。
 - d. 選擇 Inbound (入站) 標籤，然後選擇 Edit (編輯)。
 - e. 選擇 Add Rule (新增規則)。
 - f. 在新列中，輸入下列內容：

類型 (Type) : Custom (自訂) **TCP Rule**

Protocol (通訊協定) : TCP

Port Range (連接埠範圍) : 8888 (或您指派給視覺化伺服器的連接埠)

來源 : Custom IP (specify address/range)

10. 如果您想要從本機瀏覽器以視覺化方式檢視資料，請輸入下列命令以將 EC2 執行個體上轉譯的資料轉送到本機機器。

```
$ ssh -Y -L localhost:8888:localhost:8888 user_id@ec2_instance_ip
```

11. 使用執行 DLAMI 與 Conda 的 EC2 執行個體的公有 IP 或 DNS 位址，以及您為 MXBoard 開啟的連接埠，開啟 MXBoard 視覺效果的網頁：

http://127.0.0.1:8888

詳細資訊

若要進一步了解 MXBoard，請參閱 [MXBoard 網站](#)。

TensorBoard

[TensorBoard](#)使您可以直觀地檢查和解釋您的 TensorFlow 運行和圖形。它運行一個 Web 服務器，提供用於查看和與 TensorBoard 可視化交互的網頁。

TensorFlow 並 TensorBoard 與孔達 (DLAMI 與康達) 一起預先安裝了深度學習 AMI。含 Conda 的 DLAMI 也包含範例指令碼，可用 TensorFlow 來訓練 MNIST 模型，並啟用額外的記錄功能。MNIST 是手寫數字的資料庫，通常用於訓練影像辨識模型。在本教學課程中，您會使用指令碼來訓練 MNIST 模型，TensorBoard 並使用記錄檔來建立視覺效果。

主題

- [訓練 MNIST 模型並以視覺化方式進行培訓 TensorBoard](#)
- [詳細資訊](#)

訓練 MNIST 模型並以視覺化方式進行培訓 TensorBoard

以視覺化方式呈現 MNIST 模型訓練 TensorBoard

1. 使用 Conda Connect 到 DLAMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。

2. 啟動 Python 2.7 TensorFlow 環境，並使用 TensorBoard 範例指令碼瀏覽至包含資料夾的目錄：

```
$ source activate tensorflow_p27
$ cd ~/examples/tensorboard/
```

3. 執行指令碼，使用啟用的延伸記錄來訓練 MNIST 模型：

```
$ python mnist_with_summaries.py
```

指令碼會將日誌寫入 /tmp/tensorflow/mnist。

4. 傳遞日誌的位置給 tensorboard：

```
$ tensorboard --logdir=/tmp/tensorflow/mnist
```

TensorBoard 在連接埠 6006 上啟動視覺化網頁伺服器。

5. 若想從本機瀏覽器輕鬆存取，您可以將 Web 伺服器連接埠變更為連接埠 80 或其他連接埠。無論您使用哪個連接埠，都必須在 EC2 安全群組中為 DLAMI 開啟此連接埠。您也可以使用連接埠轉送。如需變更安全群組設定和連接埠轉送的詳細資訊，請參閱[設定 Jupyter 筆記本伺服器](#)。預設設定將於後續步驟中說明。

Note

如果您需要同時運行 Jupyter 服務器和服務器，請為每個 TensorBoard 服務器使用不同的端口。

6. 在 EC2 執行個體上開啟連接埠 6006 (或您指派給視覺化 Web 伺服器的連接埠)。
 - a. [在 Amazon EC2 控制台中打開您的 EC2 實例](https://console.aws.amazon.com/ec2/)，網址為 <https://console.aws.amazon.com/ec2/>。
 - b. 在 Amazon EC2 主控台中，選擇「網路和安全性」，然後選擇「安全群組」。
 - c. 對於 Security Group (安全群組)，選擇最近建立的項目 (請參考描述中的時間戳記)。
 - d. 選擇 Inbound (入站) 標籤，然後選擇 Edit (編輯)。
 - e. 選擇 Add Rule (新增規則)。
 - f. 在新資料列中輸入下列資訊：

類型 (Type) : Custom (自訂) **TCP Rule**

Protocol (通訊協定) : TCP

Port Range (連接埠範圍) : **6006** (或您指派給視覺化伺服器的連接埠)

來源 : **Custom IP (specify address/range)**

7. 使用與 Conda 執行 DLAMI 的 EC2 執行個體的公用 IP 或 DNS 位址，以及您為其開啟的連接埠，開啟 TensorBoard 視覺效果的網頁：TensorBoard

[http:// **YourInstancePublicDNS:6006**](http://YourInstancePublicDNS:6006)

詳細資訊

要了解更多信息 TensorBoard，請[TensorBoard訪問網站](#)。

分散式訓練

瞭解 DLAMI 針對多個 GPU 進行訓練時所提供的選項。若要提升效能，請參閱 [Elastic Fabric Adapter](#) 按一下其中一個選項，了解如何使用它。

主題

- [Chainer](#)
- [Keras 與 MXNet](#)
- [TensorFlow 與霍洛沃德](#)

Chainer

Note

從 v28 版本 AWS Deep Learning AMI 開始，我們不再包含 Chainer Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

[Chainer](#) 是一種靈活的 Python 型架構，讓您輕鬆、直覺地編寫複雜的神經網路架構。Chainer 可讓您輕鬆地使用多 GPU 執行個體來進行訓練。Chainer 也會自動記錄結果、將損失和準確性製成圖表，並產生輸出以供使用[運算圖](#)來視覺化呈現神經網路。它包含在孔達 (DLAMI AMI 與康達) 的深度學習 AMI 中。

下列主題說明如何在多重 GPU、單一 GPU 以及 CPU 上進行訓練、建立視覺化效果和測試您的 Chainer 安裝。

主題

- [使用 Chainer 訓練模型](#)
- [使用 Chainer 在多個 GPU 上訓練](#)
- [使用 Chainer 在單一 GPU 上訓練](#)
- [使用 Chainer 搭配 CPU 進行訓練](#)
- [繪製結果](#)
- [測試 Chainer](#)
- [詳細資訊](#)

使用 Chainer 訓練模型

本教學課程說明如何使用範例 Chainer 指令碼來訓練使用 MNIST 資料集的模型。MNIST 是手寫數字的資料庫，通常用於訓練影像辨識模型。此教學課程也會說明在 CPU 和一或多個 GPU 上訓練的訓練速度差異。

使用 Chainer 在多個 GPU 上訓練

在多個 GPU 上訓練

1. 透過 Conda Connect 至執行深度學習 AMI 的執行個體。有關如何選擇 [the section called “執行個體選擇”](#) 或連接到執行個體，請參閱或 [Amazon EC2 文件](#)。若要執行此教學課程，您將需要使用至少具備兩個 GPU 的執行個體。
2. 啟用 Python 3 Chainer 環境：

```
$ source activate chainer_p36
```

3. 若要取得最新的教學課程，請複製 Chainer 儲存庫並導覽至範例資料夾：

```
(chainer_p36) :~$ cd ~/src
(chainer_p36) :~/src$ CHAINER_VERSION=v$(python -c "import chainer;
print(chainer.__version__)")
(chainer_p36) :~/src$ git clone -b $CHAINER_VERSION https://github.com/chainer/
chainer.git
(chainer_p36) :~/src$ cd chainer/examples/mnist
```

- 執行 `train_mnist_data_parallel.py` 指令碼中的範例。根據預設，指令碼會使用在您執行深度學習 AMI 執行個體上執行的 GPU 搭配 Conda。指令碼最多可以在兩個 GPU 上執行。它會忽略超過前兩個 GPU 的任何 GPU。它會自動偵測到一個或兩個。如果您執行的是沒有 GPU 的執行個體，請跳到此教學課程稍後的[使用 Chainer 搭配 CPU 進行訓練](#)。

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist_data_parallel.py
```

Note

這個範例將會傳回因未在 DLAMI 包含 beta 版功能而引發的錯誤。

```
chainerx ModuleNotFoundError: No module named 'chainerx'
```

Chainer 指令碼使用 MNIST 資料庫來訓練模型時，您會看到每個 epoch 的結果。

然後，您會看到指令碼執行的範例輸出。以下範例輸出是在 p3.8xlarge 執行個體上執行。指令碼的輸出顯示「GPU: 0, 1」，這表示它使用四個可用 GPU 的前兩個。指令碼通常使用以零開始的 GPU 索引，而非總數。

```
GPU: 0, 1

# unit: 1000
# Minibatch-size: 400
# epoch: 20

epoch      main/loss    validation/main/loss  main/accuracy  validation/main/
accuracy  elapsed_time
1          0.277561    0.114709              0.919933      0.9654
   6.59261
2          0.0882352   0.0799204            0.973334      0.9752
   8.25162
3          0.0520674   0.0697055            0.983967      0.9786
   9.91661
4          0.0326329   0.0638036            0.989834      0.9805
  11.5767
5          0.0272191   0.0671859            0.9917         0.9796
  13.2341
6          0.0151008   0.0663898            0.9953         0.9813
  14.9068
```

7	0.0137765	0.0664415	0.995434	0.982
16.5649				
8	0.0116909	0.0737597	0.996	0.9801
18.2176				
9	0.00773858	0.0795216	0.997367	0.979
19.8797				
10	0.00705076	0.0825639	0.997634	0.9785
21.5388				
11	0.00773019	0.0858256	0.9978	0.9787
23.2003				
12	0.0120371	0.0940225	0.996034	0.9776
24.8587				
13	0.00906567	0.0753452	0.997033	0.9824
26.5167				
14	0.00852253	0.082996	0.996967	0.9812
28.1777				
15	0.00670928	0.102362	0.997867	0.9774
29.8308				
16	0.00873565	0.0691577	0.996867	0.9832
31.498				
17	0.00717177	0.094268	0.997767	0.9802
33.152				
18	0.00585393	0.0778739	0.998267	0.9827
34.8268				
19	0.00764773	0.107757	0.9975	0.9773
36.4819				
20	0.00620508	0.0834309	0.998167	0.9834
38.1389				

5. 當您的訓練仍在執行中，查看 GPU 使用率會很有用。您可以驗證哪些 GPU 處於作用中狀態，並檢視其負載。NVIDIA 提供用於此目的的工具，可使用命令 `nvidia-smi` 來執行。不過，此工具只會提供使用率快照，所以結合此工具與 Linux 命令 `watch` 能提供更多資訊。以下命令將使用 `watch` 搭配 `nvidia-smi`，每十分之一秒重新整理一次目前的 GPU 使用率。對 DLAMI 開啟另一個終端機工作階段，並執行下列命令：

```
(chainer_p36) :~$ watch -n0.1 nvidia-smi
```

您將會看到類似下列結果的輸出。使用 `ctrl-c` 關閉工具，或讓它繼續執行，同時嘗試第一個終端機工作階段中的其他範例。

```

Every 0.1s: nvidia-smi                                     Wed Feb 28 00:28:50 2018

Wed Feb 28 00:28:50 2018
+-----+
| NVIDIA-SMI 384.111                Driver Version: 384.111                |
+-----+-----+-----+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+
|   0   Tesla V100-SXM2...    On   | 00000000:00:1B.0 Off  |
| N/A   46C    P0     56W / 300W |  728MiB / 16152MiB |    10%    Default  |
+-----+-----+-----+-----+-----+-----+
|   1   Tesla V100-SXM2...    On   | 00000000:00:1C.0 Off  |
| N/A   44C    P0     53W / 300W |  696MiB / 16152MiB |     4%    Default  |
+-----+-----+-----+-----+-----+-----+
|   2   Tesla V100-SXM2...    On   | 00000000:00:1D.0 Off  |
| N/A   42C    P0     38W / 300W |   10MiB / 16152MiB |     0%    Default  |
+-----+-----+-----+-----+-----+-----+
|   3   Tesla V100-SXM2...    On   | 00000000:00:1E.0 Off  |
| N/A   46C    P0     40W / 300W |   10MiB / 16152MiB |     0%    Default  |
+-----+-----+-----+-----+-----+-----+

+-----+
| Processes:                                     GPU Memory |
|  GPU       PID    Type    Process name      Usage      |
+-----+-----+-----+-----+-----+-----+
|    0       54418    C      python             718MiB |
|    1       54418    C      python             686MiB |
+-----+-----+-----+-----+-----+-----+

```

在這個範例中，GPU 0 和 GPU 1 處於作用中狀態，GPU 2 和 3 則不是。您也可以查看每個 GPU 的記憶體使用率。

6. 訓練完成後，請記下第一個終端機工作階段的經過時間。在此範例中，經過時間是 38.1389 秒。

使用 Chainer 在單一 GPU 上訓練

此範例說明如何在單一 GPU 上訓練。如果您只有一個 GPU，或只是想了解多個 GPU 訓練搭配 Chainer 的擴展程度，便可執行此操作。

使用 Chainer 在單一 GPU 上訓練

- 在這個範例中，您使用另一個指令碼 `train_mnist.py`，並使用 `--gpu=0` 告知它只使用 GPU 0。若要在 `nvidia-smi` 主控台中查看不同的 GPU 如何啟用，您可以使用 `--gpu=1`，告訴指令碼使用 GPU 號碼 1。

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist.py --gpu=0
```

```
GPU: 0
# unit: 1000
# Minibatch-size: 100
# epoch: 20

epoch      main/loss  validation/main/loss  main/accuracy  validation/main/
accuracy  elapsed_time
1          0.192348  0.0909235            0.940934      0.9719
    5.3861
2          0.0746767  0.069854            0.976566      0.9785
    8.97146
3          0.0477152  0.0780836            0.984982      0.976
    12.5596
4          0.0347092  0.0701098            0.988498      0.9783
    16.1577
5          0.0263807  0.08851             0.991515      0.9793
    19.7939
6          0.0253418  0.0945821            0.991599      0.9761
    23.4643
7          0.0209954  0.0683193            0.993398      0.981
    27.0317
8          0.0179036  0.080285            0.994149      0.9819
    30.6325
9          0.0183184  0.0690474            0.994198      0.9823
    34.2469
10         0.0127616  0.0776328            0.996165      0.9814
    37.8693
11         0.0145421  0.0970157            0.995365      0.9801
    41.4629
12         0.0129053  0.0922671            0.995899      0.981
    45.0233
13         0.0135988  0.0717195            0.995749      0.9857
    48.6271
```

14	0.00898215	0.0840777	0.997216	0.9839
52.2269				
15	0.0103909	0.123506	0.996832	0.9771
55.8667				
16	0.012099	0.0826434	0.996616	0.9847
59.5001				
17	0.0066183	0.101969	0.997999	0.9826
63.1294				
18	0.00989864	0.0877713	0.997116	0.9829
66.7449				
19	0.0101816	0.0972672	0.996966	0.9822
70.3686				
20	0.00833862	0.0899327	0.997649	0.9835
74.0063				

在這個範例中，在單一 GPU 上執行幾乎要花上兩倍時間！訓練更大型模型或更大型資料集可能會從這個範例產生不同的結果，因此請進行實驗，以進一步評估 GPU 效能。

使用 Chainer 搭配 CPU 進行訓練

現在，請在僅限 CPU 的模式上嘗試訓練。執行相同的指令碼 `python train_mnist.py`，無需引數：

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist.py
```

在輸出中，GPU: -1 代表沒有使用 GPU：

```
GPU: -1
# unit: 1000
# Minibatch-size: 100
# epoch: 20

epoch      main/loss  validation/main/loss  main/accuracy  validation/main/accuracy
elapsed_time
1          0.192083  0.0918663             0.94195        0.9712
11.2661
2          0.0732366 0.0790055             0.977267       0.9747
23.9823
3          0.0485948 0.0723766             0.9844         0.9787
37.5275
4          0.0352731 0.0817955             0.987967       0.9772
51.6394
```


5	0.029566	0.0807774	0.990217	0.9764
65.2657				
6	0.025517	0.0678703	0.9915	0.9814
79.1276				
7	0.0194185	0.0716576	0.99355	0.9808
93.8085				
8	0.0174553	0.0786768	0.994217	0.9809
108.648				
9	0.0148924	0.0923396	0.994983	0.9791
123.737				
10	0.018051	0.099924	0.99445	0.9791
139.483				
11	0.014241	0.0860133	0.995783	0.9806
156.132				
12	0.0124222	0.0829303	0.995967	0.9822
173.173				
13	0.00846336	0.122346	0.997133	0.9769
190.365				
14	0.011392	0.0982324	0.996383	0.9803
207.746				
15	0.0113111	0.0985907	0.996533	0.9813
225.764				
16	0.0114328	0.0905778	0.996483	0.9811
244.258				
17	0.00900945	0.0907504	0.9974	0.9825
263.379				
18	0.0130028	0.0917099	0.996217	0.9831
282.887				
19	0.00950412	0.0850664	0.997133	0.9839
303.113				
20	0.00808573	0.112367	0.998067	0.9778
323.852				

在這個範例中，MNIST 在 323 秒內完成訓練，比使用兩個 GPU 進行訓練的時間超過 11 倍。如果您曾懷疑 GPU 的能力，這個範例顯示它們的效率有多高。

繪製結果

Chainer 也會自動記錄結果、將損失和準確性製成圖表，並產生輸出以供繪製運算圖。

產生運算圖

1. 任何訓練執行完成之後，您可以導覽至 `result` 目錄並在兩個自動產生的影像中檢視執行的準確度和損失。現在導覽至該處，並列出內容：

```
(chainer_p36) :~/src/chainer/examples/mnist$ cd result
(chainer_p36) :~/src/chainer/examples/mnist/result$ ls
```

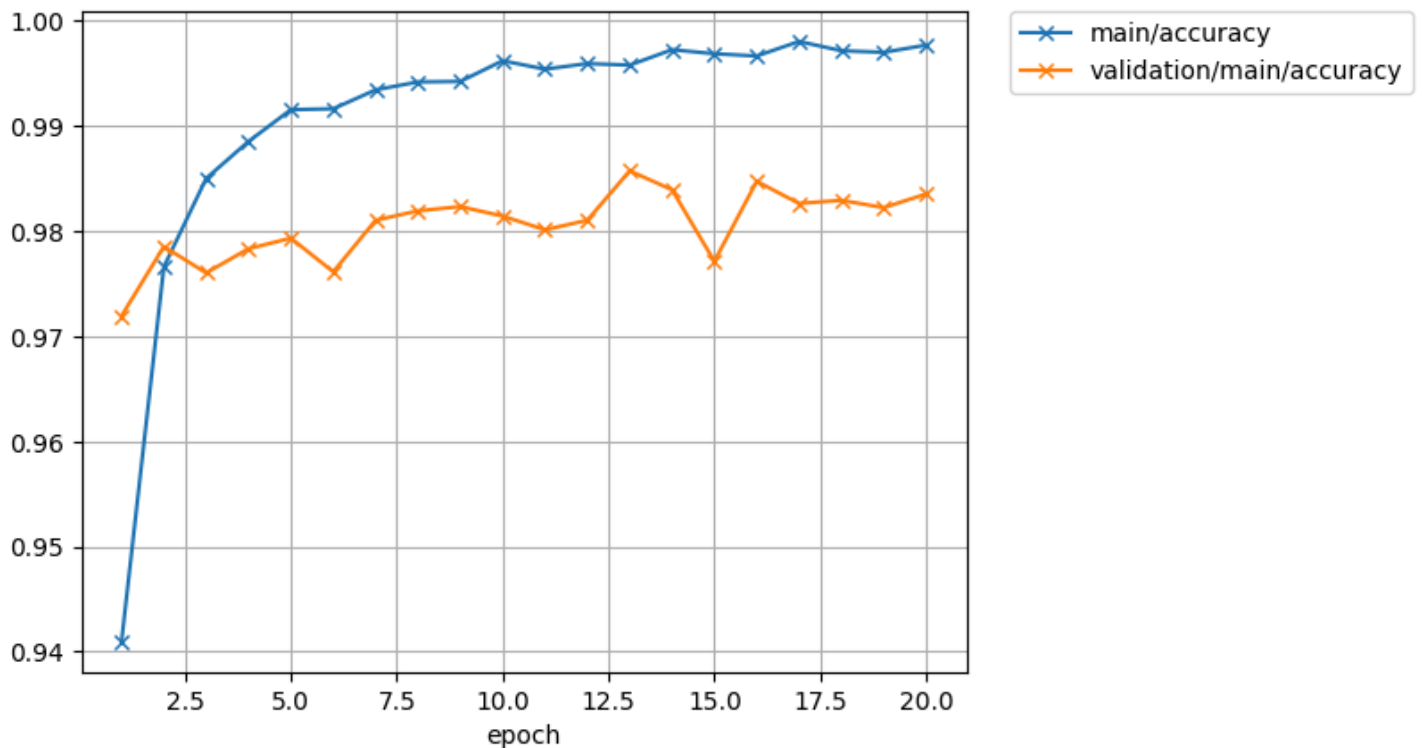
`result` 目錄包含 `.png` 格式的兩個檔案：`accuracy.png` 和 `loss.png`。

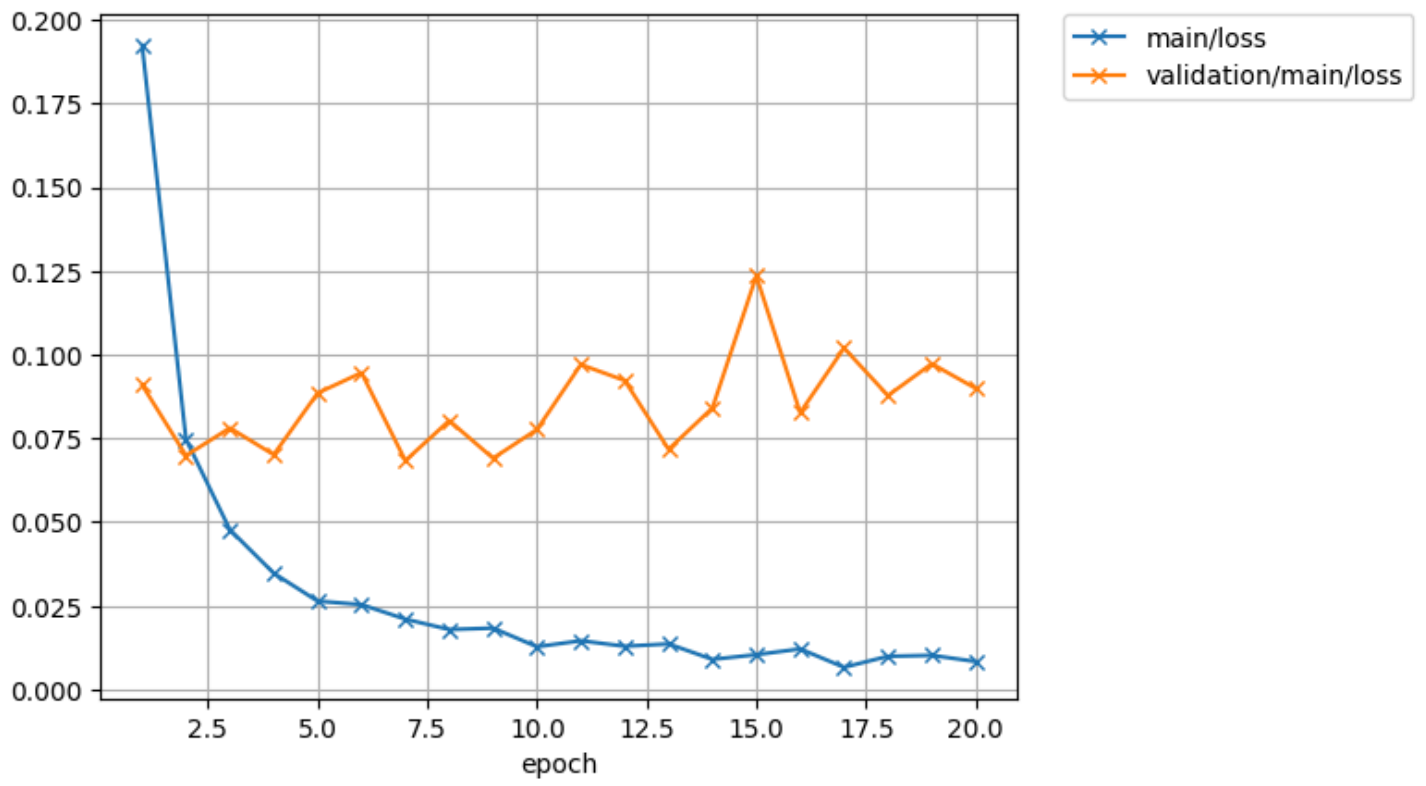
2. 若要查看圖表，請使用 `scp` 命令，將它們複製到本機電腦。

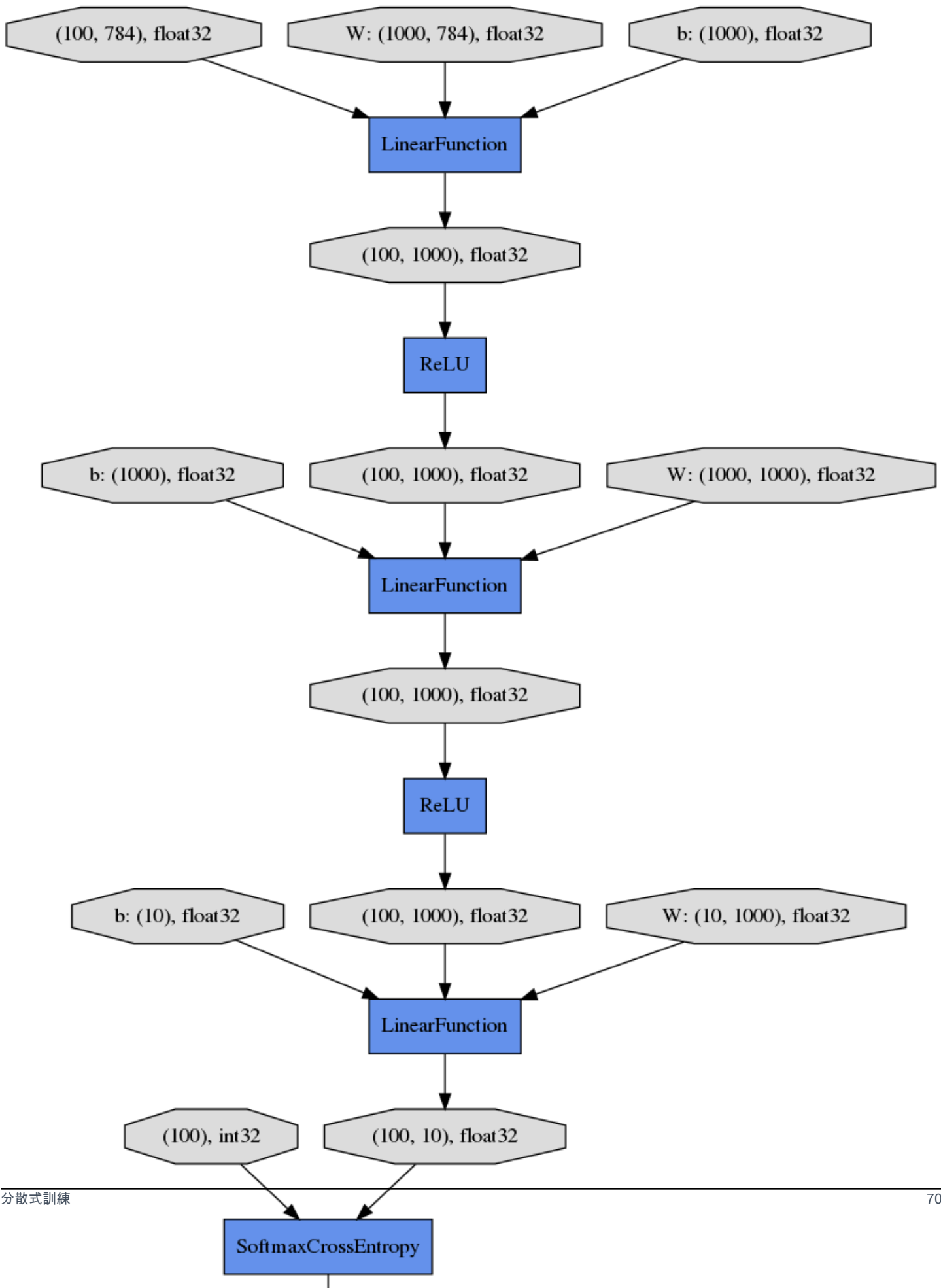
在 macOS 終端機，執行以下 `scp` 命令以將這三個檔案下載到您的 Downloads 資料夾。將金鑰檔案位置和伺服器地址的預留位置更換為您的資訊。對於其他作業系統，請使用適當的 `scp` 命令格式。請注意，對於 Amazon Linux AMI，使用者名稱為 `ec2-user`。

```
(chainer_p36) :~/src/chainer/examples/mnist/result$ scp -i "your-key-file.pem"
ubuntu@your-dlami-address.compute-1.amazonaws.com:~/src/chainer/examples/mnist/
result/*.png ~/Downloads
```

以下影像分別為準確度、損失和運算圖的範例。







測試 Chainer

若要使用預先安裝的測試指令碼來測試 Chainer 和驗證 GPU 支援，請執行下列命令：

```
(chainer_p36) :~/src/chainer/examples/mnist/result$ cd ~/src/bin
(chainer_p36) :~/src/bin$ ./testChainer
```

這會下載 Chainer 原始程式碼並執行 Chainer 多 GPU MNIST 範例。

詳細資訊

若要進一步了解 Chainer，請參閱 [Chainer 文件網站](#)。Chainer 範例資料夾包含多個範例。試試看這些範例，了解如何執行。

Keras 與 MXNet

本教學課程說明如何在具有 Conda 的深度學習 AMI 上使用 MXNet 後端啟動和使用 Keras 2。

使用 MXNet 後端啟用 Keras，並使用康達在 DLAMI 上進行測試

1. 若要使用 MXNet 後端啟用 Keras，請使用 Conda 開啟 DLAMI 的亞馬遜彈性運算雲端 (Amazon EC2) 執行個體。

- 對於 Python 3，執行此命令：

```
$ source activate mxnet_p36
```

- 對於 Python 2，執行此命令：

```
$ source activate mxnet_p27
```

2. 啟動 iPython 終端機：

```
(mxnet_p36)$ ipython
```

3. 使用 MXNet 測試匯入 Keras，確定是否正常運作：

```
import keras as k
```

您的畫面上應該會出現下列內容 (可能出現在幾個警告訊息之後)。

```
Using MXNet backend
```

Note

如果出現錯誤，或者 TensorFlow 後端仍在使用中，則需要手動更新 Keras 設定。編輯 `~/.keras/keras.json` 檔案並將後端設定變更為 `mxnet`。

Keras-MXNet 多重 GPU 訓練教學課程

訓練卷積神經網路 (CNN)

1. 打開一個終端機，然後 SSH 進入您的 DLAMI。
2. 導覽至 `~/examples/keras-mxnet/` 資料夾。
3. 在終端機視窗中執行 `nvidia-smi`，判斷 DLAMI 上可用的 GPU 數量。在接下來的步驟中，如果您有四個 GPU，您將依原樣執行指令碼。
4. (選用) 執行以下命令，開啟指令碼以進行編輯。

```
(mxnet_p36)$ vi cifar10_resnet_multi_gpu.py
```

5. (選用) 指令碼有以下行，用以定義 GPU 數。視需要更新該值。

```
model = multi_gpu_model(model, gpus=4)
```

6. 現在，執行訓練。

```
(mxnet_p36)$ python cifar10_resnet_multi_gpu.py
```

Note

使用 `channels_first` `image_data_format` 集時，Keras-MXNet 執行速度提高了兩倍。若要變更為 `channels_first`，請編輯您的 Keras 組態檔 (`~/.keras/keras.json`) 並設定下列項目：`"image_data_format": "channels_first"`。

如需調校技術的更多詳細資訊，請參閱 [Keras-MXNet 效能調校指南](#)。

詳細資訊

- 您可以在具有 `Conda ~/examples/keras-mxnet` 的深度学习 AMI 目錄中找到具有 MXNet 後端的 Keras 範例。
- 如需更多教學課程和範例，請參閱 [Keras-MX GitHub](#) Net 專案。

TensorFlow 與霍洛沃德

本教程介紹瞭如何在 TensorFlow 具有 Conda 的深度学习 AMI 上與 Horovod 一起使用。霍羅維德已預先安裝在康達環境中的。TensorFlow 建議使用 Python 3 環境。這裡的指示假設您有一個運作中 DLAMI 執行個體，其具有一個或多個 GPU。如需詳細資訊，請參閱 [如何開始使用 DLAMI](#)。

Note

只支援 P3.*、P2.* 和 G3.* 執行個體類型。

Note

`mpirun` 可從二處取得 (經由 OpenMPI)。可以在 `/usr/bin` 和 `/home/ubuntu/anaconda3/envs/<env>/bin` 中取得。`env` 是與架構對應的環境，例如 TensorFlow 和 Apache MXNet。在此類 conda 環境中，可使用版本較新的 OpenMPI。建議您使用 `mpirun` 二進位檔的絕對路徑，或 `--prefix flag` 來執行 mpi 工作負載。例如，使用 TensorFlow python36 環境時，可選用下列其中一種步驟：

```
/home/ubuntu/anaconda3/envs/tensorflow_p36/bin/mpirun <args>  
  
or  
  
mpirun --prefix /home/ubuntu/anaconda3/envs/tensorflow_p36/bin <args>
```

使用霍洛沃德啟動 TensorFlow 和測試

1. 驗證執行個體具備使用中 GPU。NVIDIA 提供下列版本適用的工具：

```
$ nvidia-smi
```

2. 激活 Python 3 TensorFlow 環境：

```
$ source activate tensorflow_p36
```

3. 啟動 iPython 終端機：

```
(tensorflow_p36)$ ipython
```

4. TensorFlow 使用 Horovod 測試匯入，以確認它是否正常運作：

```
import horovod.tensorflow as hvd  
hvd.init()
```

您的畫面上可能會出現下列內容 (可能出現在幾個警告訊息之後)。

```
-----  
[[55425,1],0]: A high-performance Open MPI point-to-point messaging module  
was unable to find any relevant network interfaces:
```

```
Module: OpenFabrics (openib)  
Host: ip-172-31-72-4
```

```
Another transport will be used instead, although this may result in  
lower performance.  
-----
```

設定您的 Horovod 主機檔案

您可以使用 Horovod 進行單一節點、多 GPU 培訓，或多節點、多 GPU 培訓。如果您計劃使用多個節點進行分散式訓練，則必須將每個 DLAMI 私有 IP 位址新增至主機檔案。您目前登錄的 DLAMI 被稱為領導者。屬於叢集一部分的其他 DLAMI 執行個體稱為成員。

在開始本節之前，請啟動一或多個 DLAMI，然後等待它們處於「就緒」狀態。範例指令碼需要一個 hosts 檔案，因此即使您打算只使用一個 DLAMI，也要建立只有一個項目的 hosts 檔案。如果您在訓練開始之後編輯主機檔案檔案，則必須重新啟動訓練，讓新增或移除的主機生效。

設定 Horovod 進行訓練

1. 切換到訓練指令碼所在的目錄。


```
cd ~/examples/horovod/tensorflow
```

2. 使用 vim 來編輯領導者主目錄中的檔案。

```
vim hosts
```

3. 在 Amazon 彈性運算雲端主控台中選取其中一個成員，主控台的說明窗格隨即顯示。尋找 Private IPs (私有 IP) 欄位，然後複製 IP 並將其貼入文字檔案中。在新行上複製每個成員的私有 IP。然後，在每個 IP 旁邊新增一個空格，接著新增文字 slots=8，如下所示。這代表每個執行個體具有多少個 GPU。p3.16xlarge 執行個體具有 8 個 GPU，因此，如果您選擇不同的執行個體類型，則將提供每個執行個體的實際 GPU 數目。對於領導者，您可以使用 localhost。有 4 個節點的叢集應該如下所示：

```
172.100.1.200 slots=8
172.200.8.99 slots=8
172.48.3.124 slots=8
localhost slots=8
```

儲存檔案並回到領導者的終端機。

4. 將成員實例使用的安全殼層金鑰新增至 ssh-agent。

```
eval `ssh-agent -s`
ssh-add <key_name>.pem
```

5. 現在您的領導者知道如何聯繫每個成員。這一切都將在私有網路界面上發生。接著，使用短 bash 函數，以協助將命令傳送至每個成員。

```
function runclust(){ while read -u 10 host; do host=${host%% slots*}; ssh -o
"StrictHostKeyChecking no" $host "$@"; done 10<$1; }
```

6. 告訴其他成員不要執行「StrickHostKeyChecking」，因為這可能會導致訓練停止回應。

```
runclust hosts "echo \"StrictHostKeyChecking no\" >> ~/.ssh/config"
```

使用合成資料進行訓練

您的 DLAMI 附帶一個示例腳本來訓練具有合成數據的模型。這會測試您的領導者是否可與叢集的成員通訊。需要主機檔案。請參閱 [設定您的 Horovod 主機檔案](#)，了解指示。

使用範例資料來測試 Horovod 訓練

1. `~/examples/horovod/tensorflow/train_synthetic.sh` 預設為 8 個 GPU，但您可以提供您想要執行的 GPU 數目。以下範例執行指令碼，以參數形式傳遞 4，表示 4 個 GPU。

```
$ ./train_synthetic.sh 4
```

在一些警告訊息之後，您看到以下輸出，確認 Horovod 正在使用 4 個 GPU。

```
PY3.6.5 |Anaconda custom (64-bit)| (default, Apr 29 2018, 16:14:56) [GCC  
7.2.0]TF1.11.0Horovod size: 4
```

然後，在一些其他警告之後，您看到資料表的開頭和一些資料點。如果您不想要監看 1,000 個批次，請細分訓練。

```
Step Epoch  Speed  Loss   FinLoss LR  
0  0.0  105.6  6.794  7.708 6.40000  
1  0.0  311.7  0.000  4.315 6.38721  
100  0.1  3010.2  0.000  34.446 5.18400  
200  0.2  3013.6  0.000  13.077 4.09600  
300  0.2  3012.8  0.000  6.196 3.13600  
400  0.3  3012.5  0.000  3.551 2.30401
```

2. Horovod 首先會使用所有本機 GPU，然後使用叢集成員的 GPU。因此，若要確定跨叢集的分散式訓練可以運作，請嘗試使用您想要使用的完整 GPU 數目。例如，如果您有 4 個成員，都是 p3.16xlarge 執行個體類型，則有 32 個跨叢集的 GPU。這是您想要嘗試完整 32 個 GPU 的位置。

```
./train_synthetic.sh 32
```

您的輸出與前一個測試類似。Horovod 大小為 32，而且速度大約快四倍。在這項實驗完成後，您已測試領導者及其與成員通訊的能力。如果您遇到任何問題，請參閱[故障診斷](#)一節。

準備資 ImageNet 料集

在本節中，您會下載 ImageNet 資料集，然後從原始資料集產生 TFRecords 格式的資料集。資料集的 DLAMI 上提供了一組預先處理指令碼，您可以將其用於其中一個 ImageNet 資料集 ImageNet 或做為其他資料集的範本。還提供了為 ImageNet 其配置的主要訓練腳本。以下部分假設您已使用具有 8 個 GPU 的 EC2 執行個體啟動 DLAMI。我們建議使用 p3.16xlarge 執行個體類型。

在 DLAMI 的~/examples/horovod/tensorflow/utils目錄中，您可以找到以下腳本：

- `utils/preprocess_imagenet.py`-使用此選項可將原始 ImageNet資料集轉換為TFRecord格式。
- `utils/tensorflow_image_resizer.py`-使用此按照建議進行 ImageNet 培訓調整數TFRecord據集的大小。

準備資 ImageNet 料集

1. 造訪 image-net.org、建立帳戶、取得存取金鑰並下載資料集。image-net.org 代管原始資料集。要下載它，您需要擁有一個 ImageNet 帳戶和一個訪問密鑰。該帳戶是免費的，要獲得免費訪問密鑰，您必須同意 ImageNet許可證。
2. 使用影像預先處理指令碼，從原始資料集產生 TFRecord 格式資料集。ImageNet 從 ~/examples/horovod/tensorflow/utils 目錄：

```
python preprocess_imagenet.py \  
    --local_scratch_dir=[YOUR DIRECTORY] \  
    --imagenet_username=[imagenet account] \  
    --imagenet_access_key=[imagenet access key]
```

3. 使用影像大小調整指令碼。如果您調整影像大小，訓練會更快速地執行，並且與[ResNet 參考 paper](#) 對齊更好。從 ~/examples/horovod/utils/preprocess 目錄：

```
python tensorflow_image_resizer.py \  
    -d imagenet \  
    -i [PATH TO TFRECORD TRAINING DATASET] \  
    -o [PATH TO RESIZED TFRECORD TRAINING DATASET] \  
    --subset_name train \  
    --num_preprocess_threads 60 \  
    --num_intra_threads 2 \  
    --num_inter_threads 2
```

在單個 DL ResNet AMI 上訓練 -50 ImageNet 模型

Note

- 本教學課程中的指令碼預期預先處理的訓練資料係位於 `~/data/tf-imagenet/` 資料夾。請參閱 [準備資 ImageNet 料集](#)，了解指示。
- 需要主機檔案。請參閱 [設定您的 Horovod 主機檔案](#)，了解指示。

使用霍羅維德在資料集上訓練 ResNet 50 個 CNN ImageNet

1. 導覽至 `~/examples/horovod/tensorflow` 資料夾。

```
cd ~/examples/horovod/tensorflow
```

2. 驗證您的組態並設定訓練中要使用的 GPU 數。首先，檢閱位於指令碼相同資料夾中的 `hosts`。如果您使用少於 8 個 GPU 的執行個體，必須更新這個檔案。此檔案預設指定 `localhost slots=8`。將數字 8 更新為您想要使用的 GPU 數。
3. 提供的 Shell 指令碼會採用您計劃使用的 GPU 數目，做為其唯一參數。執行此指令碼來開始訓練。以下範例使用 4，表示四個 GPU。

```
./train.sh 4
```

4. 它需要數小時才能完成。它使用 `mpirun` 將訓練分配在您的 GPU 間。

在 DLAMI 集群上訓練 ResNet -50 ImageNet 模型

Note

- 本教學課程中的指令碼預期預先處理的訓練資料係位於 `~/data/tf-imagenet/` 資料夾。請參閱 [準備資 ImageNet 料集](#)，了解指示。
- 需要主機檔案。請參閱 [設定您的 Horovod 主機檔案](#)，了解指示。

此範例將引導您逐步針對已準備好的資料集，在 DLAMI 叢集中的多個節點上訓練 ResNet -50 模型。

- 如需更快的效能，我們建議您在叢集的每個成員上具有本機資料集。

使用此 `copyclust` 函數將資料複製到其他成員。

```
function copyclust(){ while read -u 10 host; do host=${host%% slots*}; rsync -azv
"$2" $host:"$3"; done 10<$1; };
```

或者，如果您有座落於 S3 儲存貯體的檔案，請使用 `runclust` 函數直接將檔案下載至每個成員。

```
runclust hosts "tmux new-session -d \"export AWS_ACCESS_KEY_ID=YOUR_ACCESS_KEY &&
export AWS_SECRET_ACCESS_KEY=YOUR_SECRET && aws s3 sync s3://your-imagenet-bucket
~/data/tf-imagenet/ && aws s3 sync s3://your-imagenet-validation-bucket ~/data/tf-
imagenet/\""
```

使用可讓您一次管理多個節點的工具，可以節省大量的時間。您可以等待每個步驟，並分開管理每個執行個體，或使用工具 (例如 `tmux` 或 `畫面`)，讓您中斷連接和恢復工作階段。

複製完成之後，您已準備好開始訓練。執行指令碼，以參數形式傳遞 32，表示我們將對此執行使用 32 個 GPU。如果擔心中斷連接和終止工作階段，進而結束訓練執行，請使用 `tmux` 或類似工具。

```
./train.sh 32
```

以下輸出是您在 32 個 GPU 上執行訓練時所看到 ImageNet 的結果。三十二個 GPU 需要 90 至 110 分鐘的時間。

```
Step Epoch Speed Loss FinLoss LR
0 0.0 440.6 6.935 7.850 0.00100
1 0.0 2215.4 6.923 7.837 0.00305
50 0.3 19347.5 6.515 7.425 0.10353
100 0.6 18631.7 6.275 7.173 0.20606
150 1.0 19742.0 6.043 6.922 0.30860
200 1.3 19790.7 5.730 6.586 0.41113
250 1.6 20309.4 5.631 6.458 0.51366
300 1.9 19943.9 5.233 6.027 0.61619
350 2.2 19329.8 5.101 5.864 0.71872
400 2.6 19605.4 4.787 5.519 0.82126
...
13750 87.9 19398.8 0.676 1.082 0.00217
13800 88.2 19827.5 0.662 1.067 0.00156
13850 88.6 19986.7 0.591 0.997 0.00104
```

```

13900  88.9 19595.1  0.598  1.003 0.00064
13950  89.2 19721.8  0.633  1.039 0.00033
14000  89.5 19567.8  0.567  0.973 0.00012
14050  89.8 20902.4  0.803  1.209 0.00002
Finished in 6004.354426383972

```

在訓練執行完成之後，指令碼接著會進行評估執行。它會在領導者上執行，因為其執行速度足夠快，以致無需將工作分配給其他成員。以下為評估執行的輸出。

```

Horovod size: 32
Evaluating
Validation dataset size: 50000
[ip-172-31-36-75:54959] 7 more processes have sent help message help-btl-vader.txt /
cma-permission-denied
[ip-172-31-36-75:54959] Set MCA parameter "orte_base_help_aggregate" to 0 to see all
help / error messages
  step  epoch  top1    top5    loss  checkpoint_time(UTC)
14075  90.0  75.716  92.91   0.97  2018-11-14 08:38:28

```

以下是此指令碼搭配 256 個 GPU 執行時的範例輸出，執行時間介於 14 分鐘與 15 分鐘之間。

```

Step Epoch  Speed  Loss  FinLoss LR
1400  71.6 143451.0  1.189  1.720 0.14850
1450  74.2 142679.2  0.897  1.402 0.10283
1500  76.7 143268.6  1.326  1.809 0.06719
1550  79.3 142660.9  1.002  1.470 0.04059
1600  81.8 143302.2  0.981  1.439 0.02190
1650  84.4 144808.2  0.740  1.192 0.00987
1700  87.0 144790.6  0.909  1.359 0.00313
1750  89.5 143499.8  0.844  1.293 0.00026
Finished in 860.5105031204224

Finished evaluation
1759  90.0  75.086  92.47   0.99  2018-11-20 07:18:18

```

故障診斷

以下命令可以協助您不理會在使用 Horovod 進行實驗時顯示的錯誤。

- 如果訓練由於某種原因而當機毀，則 mpirun 可能無法在每部機器上清除所有 python 程序。在這種情況下，在開始下一個作業之前，請停止所有計算機上的 python 進程，如下所示：

```
runclust hosts "pkill -9 python"
```

- 如果程序突然完成，但沒有發生錯誤，請嘗試刪除您的日誌資料夾。

```
runclust hosts "rm -rf ~/imagenet_resnet/"
```

- 如果冒出其他無法解釋的問題，請檢查您的磁碟空間。如果當掉，請嘗試移除日誌資料夾，因為那裡充滿檢查點和資料。您也可以增加每個成員的磁碟區大小。

```
runclust hosts "df /"
```

- 最後一個方法為您也可以嘗試重新開機。

```
runclust hosts "sudo reboot"
```

如果您嘗試在不支援的執行個體類型上 TensorFlow 搭配 Horovod 使用，可能會收到下列錯誤碼：

```
-----  
NotFoundError Traceback (most recent call last)  
<ipython-input-3-e90ed6cabab4> in <module>()  
----> 1 import horovod.tensorflow as hvd  
  
~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/  
__init__.py in <module>()  
** *34* check_extension('horovod.tensorflow', 'HOROVOD_WITH_TENSORFLOW', __file__,  
    'mpi_lib')  
** *35*  
---> 36 from horovod.tensorflow.mpi_ops import allgather, broadcast, _allreduce  
** *37* from horovod.tensorflow.mpi_ops import init, shutdown  
** *38* from horovod.tensorflow.mpi_ops import size, local_size, rank, local_rank  
  
~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/  
mpi_ops.py in <module>()  
** *56*  
** *57* MPI_LIB = _load_library('mpi_lib' + get_ext_suffix(),  
---> 58 ['HorovodAllgather', 'HorovodAllreduce'])  
** *59*  
** *60* _basics = _HorovodBasics(__file__, 'mpi_lib')
```

```
~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/
mpi_ops.py in _load_library(name, op_list)
** *43* """
** *44* filename = resource_loader.get_path_to_datafile(name)
--> 45 library = load_library.load_op_library(filename)
** *46* for expected_op in (op_list or []):
** *47* for lib_op in library.OP_LIST.op:

~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/tensorflow/python/
framework/load_library.py in load_op_library(library_filename)
** *59* RuntimeError: when unable to load the library or get the python wrappers.
** *60* """
--> 61 lib_handle = py_tf.TF_LoadLibrary(library_filename)
** *62*
** *63* op_list_str = py_tf.TF_GetOpList(lib_handle)

NotFoundError: /home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/
horovod/tensorflow/mpi_lib.cpython-36m-x86_64-linux-gnu.so: undefined symbol:
_ZN10tensorflow14kernel_factory17OpKernelRegistrar12InitInternalEPKNS_9KernelDefEN4abs111string
```

詳細資訊

如需公用程式和範例，請參閱 DLAMI 主目錄中的~/examples/horovod資料夾。

如需更多教學課程和範例，請參閱 [Horovod GitHub](#) 專案。

Elastic Fabric Adapter

[彈性網狀架構介面卡 \(EFA\)](#) 是一種網路裝置，您可以連接至 DLAMI 執行個體，以加速高效能運算 (HPC) 應用程式。EFA 可讓您透過雲端提供的延展性、彈性和彈性，達到內部部署 HPC 叢集的應用程式效能。AWS

下列主題說明如何開始使用 EFA 搭配 DLAMI。

Note

從此[基本 GPU Diami](#) 列表中選擇您的 DLAMI

主題

- [使用 EFA 啟動 AWS Deep Learning AMI 執行個體](#)
- [使用全民福利局於全港特別行政區](#)

使用 EFA 啟動 AWS Deep Learning AMI 執行個體

最新的基礎 DLAMI 已準備好與 EFA 搭配使用，並隨附適用於 GPU 執行個體的必要驅動程式、核心模組、libFabric、openmpi 和 [N CCL OFI](#) 外掛程式。

[您可以在發行說明中找到支援的基礎 DLAMI 的 CUDA 版本。](#)

請注意：

- mpirun 在 EFA 上使用執行 NCCL 應用程式時，您必須將 EFA 支援安裝的完整路徑指定為：

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

- 若要讓您的應用程式能夠使用 EFA，請新增 FI_PROVIDER="efa" 至 mpirun 命令，如 [使用全民福利局於全港特別行政區](#) 中所示。

主題

- [準備已啟用 EFA 的安全性群組](#)
- [啟動您的執行個體](#)
- [驗證 EFA 附件](#)

準備已啟用 EFA 的安全性群組

EFA 需要一個安全性群組，允許進出安全性群組本身的所有輸入和輸出流量。如需詳細資訊，請參閱 [EFA 文件](#)。

1. 在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇 Security Groups (安全群組)，然後選擇 Create Security Group (建立安全群組)。
3. 在 Create Security Group (建立安全群組) 視窗中，執行下列動作：
 - 對於 Security group name (安全群組名稱)，輸入安全群組的描述性名稱，例如 EFA-enabled security group。
 - (選用) 對於 Description (描述)，輸入安全群組的簡短描述。
 - 對於 VPC，選取您打算讓具備 EFA 功能的執行個體在其中啟動的 VPC。
 - 選擇 Create (建立)。

4. 選取您建立的安全群組，在 Description (描述) 索引標籤上，複製 Group ID (群組 ID)。
5. 在「入埠」和「出埠」索引標籤上，執行下列動作：
 - 選擇 Edit (編輯)。
 - 針對 Type (類型)，選擇 All traffic (所有流量)。
 - 對於 Source (資源)，選擇 Custom (自訂)。
 - 將您複製的安全群組 ID 貼到欄位中。
 - 選擇儲存。
6. 參照 [授權 Linux 執行個體的傳入流量](#) 來啟用傳入流量。如果跳過此步驟，將無法與 DLAMI 執行個體進行通訊。

啟動您的執行個體

上的 EFA 目前支援下列執行個體類型和作業系統：AWS Deep Learning AMI

- 全天候:Amazon Linux 2, Ubuntu
- 大型:Amazon Linux 2
- 大:Amazon Linux 2,

下一節說明如何啟動已啟用 EFA 的 DLAMI 執行個體。如需啟動 EFA 啟用執行個體的詳細資訊，請參閱 [將啟用 EFA 的執行個體啟動至叢集置放群組](#)。

1. 在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 選擇 Launch Instance (啟動執行個體)。
3. 在「選擇 AMI」頁面上，選取 DLAMI 版本說明頁面上支援的 [DLAMI](#)。
4. 在 [選擇執行個體類型] 頁面上，選取下列其中一個支援的執行個體類型，然後選擇下一步：設定執行個體詳細資訊 如需支援的執行個體清單，請參閱 [此連結：開始使用 EFA 和 MPI](#)
5. 在 Configure Instance Details (設定執行個體詳細資訊) 頁面上，執行下列操作：
 - 對於 Number of instances (執行個體的數目)，輸入要啟動的具備 EFA 功能的執行個體數。
 - 對於 Network (網路) 和 Subnet (子網)，選取要在其中啟動執行個體的 VPC 和子網。
 - [選擇性] 對於「放置」群組，選取「將例證新增至放置群組」。為獲得最佳效能，請在置放群組內啟動執行個體。
 - [選擇性] 在「放置群組名稱」中，選取「新增至新的放置群組」，輸入放置群組的描述性名稱，然後針對「放置群組策略」選取叢集。

- 確保在此頁面上啟用「彈性織物適配器」。如果停用此選項，請將子網路變更為支援您所選執行個體類型的子網路。
 - 在 Network Interfaces (網路介面) 區段中，針對裝置 eth0，選擇 New network interface (新網路介面)。您可以選擇性指定一個主要 IPv4 地址，以及一或多個次要 IPv4 地址。如果您在有相關聯 IPv6 CIDR 區塊的子網中啟動執行個體，您可以選擇性指定一個主要 IPv6 地址，以及一或多個次要 IPv6 地址。
 - 選擇 Next: Add Storage (下一步：新增儲存體)。
6. 在 Add Storage (新增儲存體) 頁面上，除了 AMI 指定的磁碟區 (例如根設備磁碟區)，指定要連接到執行個體的磁碟區，然後選擇 Next: Add Tags (下一步：新增標籤)。
 7. 在 Add Tags (新增標籤) 頁面上，為執行個體指定標籤 (例如使用者易記的名稱)，然後選擇 Next: Configure Security Group (下一步：設定安全群組)。
 8. 在 [設定安全性群組] 頁面上，對於 [指派安全性群組]，選取 [選取現有的安全性群組]，然後選取先前建立的安全性群組。
 9. 選擇 Review and Launch (檢閱和啟動)。
 10. 在 Review Instance Launch (檢閱執行個體啟動) 頁面上，檢閱設定，然後選擇 Launch (啟動)，以選擇金鑰對並啟動執行個體。

驗證 EFA 附件

從主控台

啟動執行個體之後，請在 AWS 主控台中檢查執行個體詳細資料。若要執行此操作，在 EC2 主控台中選取執行個體，然後查看頁面下方窗格中的 [Description (描述)] 索引標籤。尋找參數“網路界面：eth0”，然後按一下 eth0 開啟一個彈出式畫面。確保「彈性織物適配器」已啟用。

如果未啟用 EFA，您可以透過以下任一方式修正此問題：

- 終止 EC2 執行個體並按照相同的步驟啟動新的執行個體。請確定已連接 EFA。
- 將 EFA 連接至現有的執行個體。
 1. 在 EC2 主控台中，移至 [Network Interfaces (網路界面)]。
 2. 按一下 [Create a Network Interface (建立網路界面)]。
 3. 選取您的執行個體所在的相同子網路。
 4. 確保啟用「彈性織物適配器」，然後單擊創建。
 5. 返回 [EC2 Instances (EC2 執行個體)] 索引標籤並選取您的執行個體。

- 移至「動作：執行個體狀態」，並在連接 EFA 之前停止執行個體。
- 從 [Actions (動作)] 中，選取 [Networking: Attach Network Interface (聯網：連接網路界面)]。
- 選擇您剛建立的界面，然後按一下連接。
- 重新啟動您的執行個體。

從執行個體

DLAMI 上已經存在以下測試腳本。執行它以確保核心模組已正確載入。

```
$ fi_info -p efa
```

您的輸出應該類似以下內容：

```
provider: efa
  fabric: EFA-fe80::e5:56ff:fe34:56a8
  domain: efa_0-rdm
  version: 2.0
  type: FI_EP_RDM
  protocol: FI_PROTO_EFA
provider: efa
  fabric: EFA-fe80::e5:56ff:fe34:56a8
  domain: efa_0-dgrm
  version: 2.0
  type: FI_EP_DGRAM
  protocol: FI_PROTO_EFA
provider: efa;ofi_rxd
  fabric: EFA-fe80::e5:56ff:fe34:56a8
  domain: efa_0-dgrm
  version: 1.0
  type: FI_EP_RDM
  protocol: FI_PROTO_RXD
```

確認安全群組組態

DLAMI 上已經存在以下測試腳本。執行它以確保您建立的安全群組已正確設定。

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

您的輸出應該類似以下內容：

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

如果它停止回應或未完成，請確定您的安全性群組具有正確的入站/輸出規則。

使用全民福利局於全港特別行政區

下節說明如何使用 EFA 在上執行多節點應用程式。 AWS Deep Learning AMI

使用 EFA 執行多節點應用程式

要跨節點集群運行應用程序，需要以下配置

主題

- [啟用無密碼 SSH](#)
- [建立主機檔案](#)
- [NCCL 測試](#)

啟用無密碼 SSH

在叢集中選取一個節點做為領導節點。其餘的節點稱為成員節點。

1. 在領導節點上，產生 RSA 金鑰對。

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

2. 變更領導節點上私有金鑰的許可。

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

3. 將公開金鑰複製到叢集中成員節點，並將其附加至叢集中的成員節點。
4. 您現在應該可以使用私有 IP 直接從領導節點登入到成員節點。

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

5. 透過將下列項目新增至引線節點上的 `~/.ssh/config` 檔案，以停用 `strictHostKey` 檢查並啟用前導節點上的代理程式轉送：

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

6. 在 Amazon Linux 2 執行個體上，在領導節點上執行下列命令，為設定檔提供正確的許可：

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

建立主機檔案

在領導節點上，建立主機檔案以識別叢集中的節點。主機檔案對於叢集中每個節點都必須有項目。建立一個檔案 `~/hosts`，並使用私有 IP 新增每個節點，如下所示：

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

NCCL 測試

Note

這些測試已使用 EFA 版本 1.30.0 和 OFI NCCL 外掛程式 1.7.4 執行。

下面列出了 Nvidia 提供的 NCCL 測試子集，用於在多個計算節點上測試功能和性能

支援的執行個體：P3dn、P4、P5

功能測試

NCCL 訊息傳輸多節點測試

`nccl_message_transfer` 是一項簡單的測試，可確保 NCCL OFI 外掛程式如預期般運作。該測試驗證了 NCCL 的連接建立和資料傳輸 API 的功能。使用 EFA 執行 NCCL 應用程式時，請確定您使用完整的執

行路徑，如範例所示。根據叢集中的執行個體和 GPU 的數目來變更參數 np 和 N。如需詳細資訊，請參閱 [AWS OFI NCC L](#) 文件。

下列 nccl_ 訊息傳輸測試適用於通用的 CUDA xx.x 版本。您可以取代指令碼中的 CUDA 版本，在 Amazon EC2 執行個體中針對任何可用的 CUDA 版本執行命令。

```
$/opt/amazon/openmpi/bin/mpirun -n 2 -N 1 --hostfile hosts \  
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/  
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:$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
```

您的輸出看起來應該如下所示。您可以檢查輸出，看看 EFA 正在用作 OFI 提供者。

```
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
```

性能測試

大型多節點非 CCL 效能測試

[要使用 EFA 檢查 NCCL 性能，請運行官方 NCCL 測試存儲庫上提供的標準 NCCL 性能測試。](#) DLAMI 隨附這個已經為 CUDA XX.X 建立的測試，您也可以使用 EFA 來執行自己的指令碼。

建構您自己的指令碼時，請參閱下列指引：

- 使用 EFA 執行 NCCL 應用程式時，請使用範例所示的完整路徑來執行完整的執行路徑。
- 根據叢集中的執行個體和 GPU 的數目來變更參數 np 和 N。
- 新增 NCCL_DEBUG = 資訊旗標，並確定記錄檔會將 EFA 用法指示為「選取的提供者為 EFA」。
- 設定要剖析的訓練記錄位置以進行驗證

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

在任何成員節點上使用命令 `watch nvidia-smi` 來監視 GPU 使用量。下列 `watch nvidia-smi` 指令適用於一般 CUDA xx.x 版本，且視執行個體的作業系統而定。您可以取代指令碼中的 CUDA 版本，在 Amazon EC2 執行個體中針對任何可用的 CUDA 版本執行命令。

- Amazon Linux 2 :

```
$ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \  
-x NCCL_DEBUG=INFO -x --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 -x NCCL_PROTO=simple -b 8 -e  
1G -f 2 -g 1 -c 1 -n 100 | tee ${TRAINING_LOG}
```

- UBUNTU

```
$ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \  
-x NCCL_DEBUG=INFO -x --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 -x NCCL_PROTO=simple-b 8 -e
1G -f 2 -g 1 -c 1 -n 100 | tee ${TRAINING_LOG}
```

您的輸出看起來應如以下所示：

```
# 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-----

				out-of-place						
#	in-place									
#	size	count	type	redop	root	time	algbw	busbw	#wrong	
#	time	algbw	busbw	#wrong						
	(us)	(B)	(elements)			(us)	(GB/s)	(GB/s)		
	0		0	float	sum	-1	11.02	0.00	0.00	0
11.04	0.00	0.00	0	float	sum	-1	11.01	0.00	0.00	0
	0		0	float	sum	-1	11.02	0.00	0.00	0
11.00	0.00	0.00	0	float	sum	-1	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	float	sum	-1	11.02	0.00	0.00	0
	0		0	float	sum	-1	11.02	0.00	0.00	0
11.02	0.00	0.00	0	float	sum	-1	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	float	sum	-1	627.4	0.00	0.00	0
	512		8	float	sum	-1	627.4	0.00	0.00	0
629.6	0.00	0.00	0	float	sum	-1	627.4	0.00	0.00	0
	1024		16	float	sum	-1	632.2	0.00	0.00	0
631.7	0.00	0.00	0	float	sum	-1	632.2	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

```

驗證測試

若要驗證 EFA 測試傳回有效的結果，請使用下列測試來確認：

- 使用 EC2 執行個體中繼資料取得執行個體類型

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

- 執行 [性能測試](#)
- 設定下列參數

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- 驗證結果，如下所示：

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

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

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

    # Validation of logs
    grep "NET/OFI Using CUDA runtime version ${CUDA_RUNTIME_VERSION}" ${TRAINING_LOG}
    || { echo "Runtime cuda text not found"; exit 1; }
    grep "NET/OFI Initializing aws-ofi-nccl" ${TRAINING_LOG} || { echo "aws-ofi-nccl
    is not working, please check if it is installed correctly"; exit 1; }
```

```

grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-
specific options text not found"; exit 1; }
grep "Using network AWS Libfabric" ${TRAINING_LOG} || { echo "AWS Libfabric text
not found"; exit 1; }
grep "busbw" ${TRAINING_LOG} || { echo "busbw text not found"; exit 1; }
grep "Avg bus bandwidth " ${TRAINING_LOG} || { echo "Avg bus bandwidth text not
found"; exit 1; }
grep "NCCL version $NCCL_VERSION" ${TRAINING_LOG} || { echo "Text not found: NCCL
version $NCCL_VERSION"; exit 1; }

if [[ ${INSTANCE_TYPE} == "p4d.24xlarge" ]]; then
    grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found:
NET/AWS Libfabric/0/GDRDMA"; exit 1; }
    grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
{ echo "Selected Provider is efa text not found"; exit 1; }
    grep "aws-ofi-nccl/xml/p4d-24xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
bandwidth text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
{ echo "Avg bus bandwidth text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p4de-24xl-topo.xml" ${TRAINING_LOG} || { echo
"Topology file not found"; exit 1; }
        elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
            grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
bandwidth text not found"; exit 1; }
            grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
{ echo "Avg bus bandwidth text not found"; exit 1; }
            grep "aws-ofi-nccl/xml/p5.48xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
file not found"; exit 1; }
            elif [[ ${INSTANCE_TYPE} == "p3dn.24xlarge" ]]; then
                grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
{ echo "Selected Provider is efa text not found"; exit 1; }
            fi
            echo "***** check_efa_nccl_all_reduce passed for cuda
version ${CUDA_VERSION} *****"
        else
            echo "***** check_efa_nccl_all_reduce failed for cuda
version ${CUDA_VERSION} *****"
        fi
    fi

```

- 要訪問基準數據，我們可以解析多節點 all_reduce 測試中表輸出的最後一行：

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

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

GPU 監控和最佳化

下一節將引導您完成 GPU 最佳化和監控選項。本節的編排像一般的工作流程，包括監控、監督、預先處理和培訓。

- [監控](#)
 - [使用以下方式監控 GPU CloudWatch](#)
- [最佳化](#)
 - [預處理](#)
 - [培訓](#)

監控

您的 DLAMI 預先安裝了幾個 GPU 監視工具。本指南還提及可供下載和安裝的工具。

- [使用以下方式監控 GPU CloudWatch](#)-預先安裝的公用程式，可向 Amazon CloudWatch 報告 GPU 使用量統計資料。
- [nvidia-smi CLI](#) - 用於監控整體 GPU 運算和記憶體使用率的公用程式。這已預先安裝在您的 AWS Deep Learning AMI (DLAMI) 上。
- [NVML C 程式庫](#) - 以 C 為基礎的 API，可直接存取 GPU 監控和管理功能。這是由 nvidia-smi CLI 在幕後使用，並預先安裝在 DLAMI 上。它還有 Python 和 Perl 繫結，有助於以這些語言來開發。預先安裝在您的 DLAMI 上的 gpumon.py 公用程式會使用來自 [nvidia-ml-py](#)
- [NVIDIA DCGM](#) - 叢集管理工具。造訪開發人員頁面，了解如何安裝和設定這個工具。

i Tip

查看 NVIDIA 的開發人員部落格，瞭解如何使用安裝 DLAMI 的 CUDA 工具的最新資訊：

- [使 TensorCore 用 Nsight IDE 和 Nvprof 監控](#)使用率。

使用以下方式監控 GPU CloudWatch

當您使用 DLAMI 搭配 GPU 時，您可能會發現您在培訓或推論期間設法追蹤其使用狀況。這在最佳化資料管道和調校深度學習網路時可能很有用。

設定 GPU 指標的方式有兩種 CloudWatch：

- [使用 AWS CloudWatch 代理程式設定測量結果 \(建議\)](#)
- [使用預先安裝gpumon.py的指令碼設定度量](#)

使用 AWS CloudWatch 代理程式設定測量結果 (建議)

將 DLAMI 與[統一 CloudWatch 代理程式](#)整合，以設定 GPU 指標，並監控 Amazon EC2 加速執行個體中 GPU 協同處理程序的使用率。

使用 DLAMI 設定 [GPU 指標](#)的方式有四種：

- [設定最低 GPU 指標](#)
- [設定部分 GPU 指標](#)
- [設定所有可用的 GPU 指標](#)
- [設定自訂 GPU 指標](#)

如需更新和安全性修補程式的資訊，請參閱 [AWS CloudWatch 代理程式的安全性修補](#)

必要條件

若要開始使用，您必須設定允許執行個體推送指標的 Amazon EC2 執行個體 IAM 許可 CloudWatch。如需詳細步驟，請參閱[建立 IAM 角色和使用者以搭配 CloudWatch 代理程式](#)使用。

設定最低 GPU 指標

使用dlami-cloudwatch-agent@minimalsystemd服務設定最低 GPU 指標。此服務會設定下列測量結果：

- utilization_gpu
- utilization_memory

您可以在下列位置找到最低預先設定 GPU 指標的systemd服務：

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

使用以下命令啟用並啟動systemd服務：

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

設定部分 GPU 指標

使用dlami-cloudwatch-agent@partialsystemd服務設定部分 GPU 指標。此服務會設定下列測量結果：

- utilization_gpu
- utilization_memory
- memory_total
- memory_used
- memory_free

您可以在下列位置找到部分預先設定 GPU 指標的systemd服務：

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

使用以下命令啟用並啟動systemd服務：

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

設定所有可用的 GPU 指標

使用dlami-cloudwatch-agent@allsystemd服務設定所有可用的 GPU 指標。此服務會設定下列測量結果：

- utilization_gpu
- utilization_memory
- memory_total
- memory_used
- memory_free
- temperature_gpu
- power_draw
- fan_speed
- pcie_link_gen_current
- pcie_link_width_current
- encoder_stats_session_count
- encoder_stats_average_fps
- encoder_stats_average_latency
- clocks_current_graphics
- clocks_current_sm
- clocks_current_memory
- clocks_current_video

您可以在下列位置找到所有可用預先設定 GPU 指標的systemd服務：

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

使用以下命令啟用並啟動systemd服務：

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

設定自訂 GPU 指標

如果預先設定的測量結果不符合您的需求，您可以建立自訂 CloudWatch 代理程式組態檔。

建立自訂規劃檔

若要建立自訂組態檔，請參閱[手動建立或編輯 CloudWatch 代理程式組態檔](#)中的詳細步驟。

在此範例中，假設結構定義位於 `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`。

使用自訂檔案設定指標

執行下列命令，根據您的自訂檔案設定 CloudWatch 代理程式：

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

AWS CloudWatch 代理程式的安全性修補

新發行的 DLAMI 已設定為最新可用的 AWS CloudWatch 代理程式安全性修補程式。請參閱以下各節，根據您選擇的作業系統，使用最新的安全性修補程式來更新您目前的 DLAMI。

Amazon Linux 2

用於取yum得適用於 Amazon Linux 2 DLAMI 的最新 AWS CloudWatch 代理程式安全修補程式。

```
sudo yum update
```

Ubuntu

若要透過 Ubuntu 取得 DLAMI 的最新 AWS CloudWatch 安全性修補程式，您必須使用 Amazon S3 下載連結重新安裝 AWS CloudWatch 代理程式。

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

如需使用 Amazon S3 下載連結安裝 AWS CloudWatch 代理程式的詳細資訊，請參閱在[伺服器上安裝和執行 CloudWatch 代理程式](#)。

使用預先安裝 `gpumon.py` 的指令碼設定度量

稱為 `gpumon.py` 的公用程式已預先安裝在 DLAMI 上。它整合 CloudWatch 並支援監控每個 GPU 的使用情況：GPU 記憶體、GPU 溫度和 GPU 電源。指令碼會定期將受監視的資料傳送至 CloudWatch。您可以透 CloudWatch 過變更指令碼中的一些設定來設定要傳送至的資料粒度層級。不過，在啟動指令碼之前，您必須先設定 CloudWatch 才能接收指標。

如何設置和運行 GPU 監控 CloudWatch

1. 建立 IAM 使用者，或修改現有的使用者，以取得將指標發佈到的政策 CloudWatch。如果您建立新的使用者，請記下登入資料，因為您在下一步驟中需要用到。

要搜尋的身分與存取權管理政策為「雲觀察：PutMetricData」。新增的政策如下所示：

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

Tip

如需建立 IAM 使用者和新增政策的詳細資訊 CloudWatch，請參閱 [CloudWatch 文件](#)。

2. 在 DLAMI 上，執行 [AWS 設定](#) 並指定 IAM 使用者登入資料。

```
$ aws configure
```

3. 執行 gpumon 公用程式之前，您可能需要對它進行一些修改。您可以在下列程式碼區塊中定義的位置找到 gpumon 公用程式和 README。如需 gpumon.py 指令碼的詳細資訊，請參閱 [指令碼的 Amazon S3 位置](#)。

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

選項：

- 如果您的執行個體「不在」us-east-1 中，請在 gpumon.py 中變更區域。
- 使用變更其他參數，例如 CloudWatchnamespace 或報表期間 store_reso。

- 此指令碼目前僅支援 Python 3。激活您首選框架的 Python 3 環境或激活 DLAMI 通用 Python 3 環境。

```
$ source activate python3
```

- 在背景執行 gpumon 公用程式。

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

- 打開瀏覽器到 <https://console.aws.amazon.com/cloudwatch/>，然後選擇指標。它將有一個命名空間 "DeepLearningTrain"。

Tip

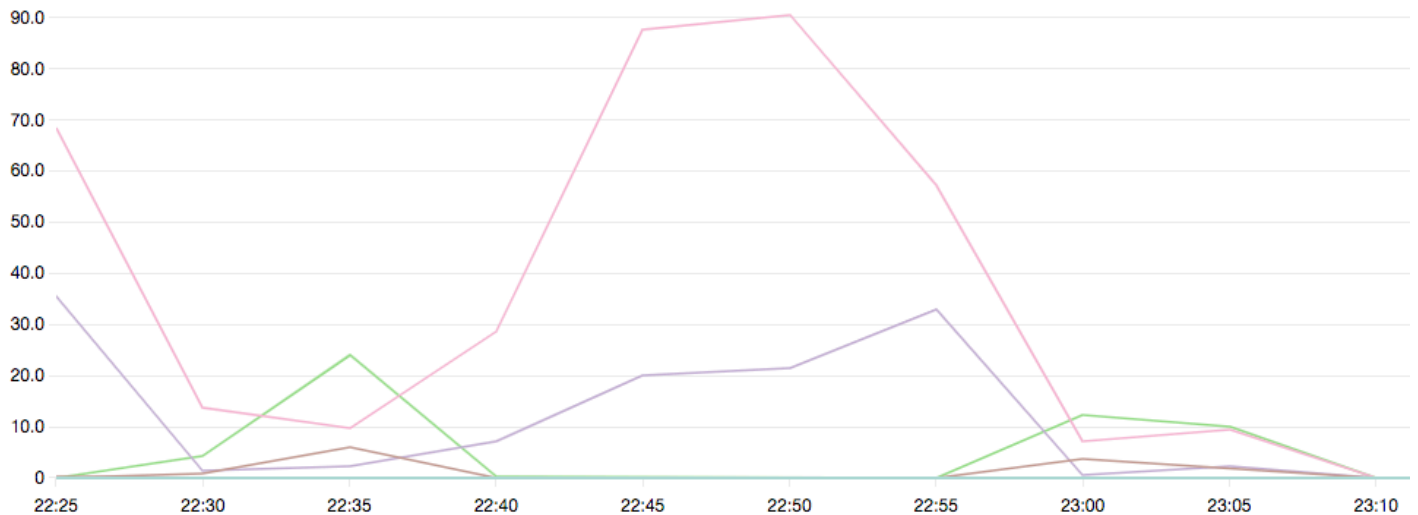
您可以修改 gpumon.py 來變更命名空間。您也可以調整 store_reso 來修改報告間隔。

以下是針對執行 gpumon.py 監視 p2.8xlarge 執行個體上訓練工作的報告範例 CloudWatch 圖表。

GPU usage, Memory usage 

1h 3h 12h 1d 3d 1w custom

Various units



對於 GPU 監控和最佳化，您可能有興趣看其他這些主題：

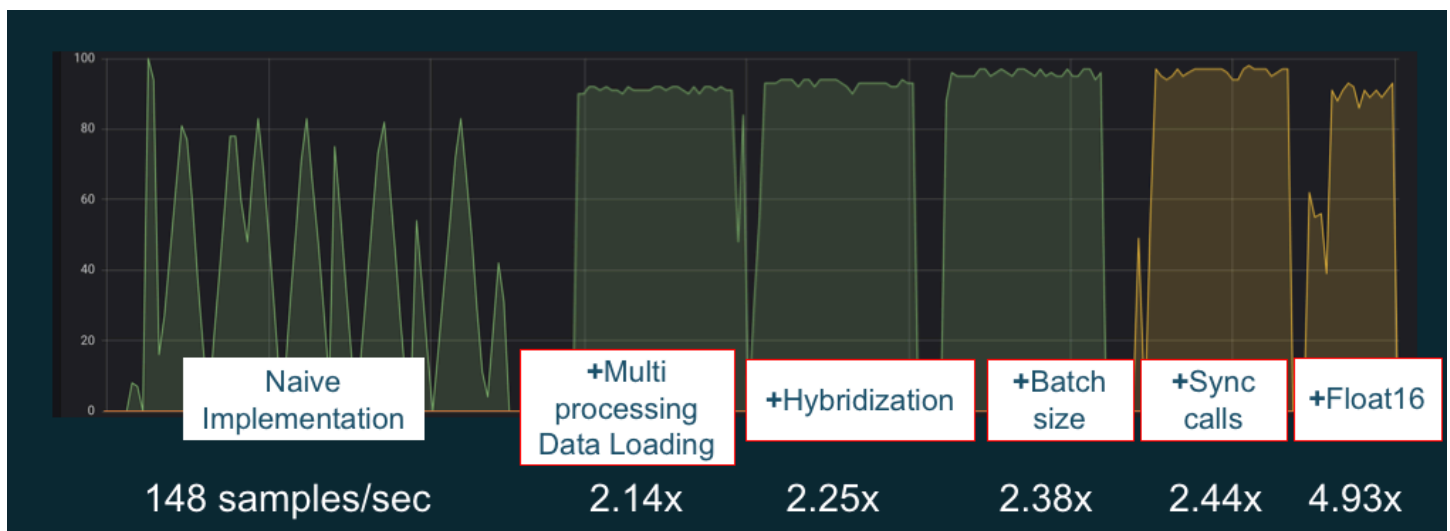
- [監控](#)
 - [使用以下方式監控 GPU CloudWatch](#)
- [最佳化](#)

- [預處理](#)
- [培訓](#)

最佳化

為了充分利用您的 GPU，您可以最佳化資料管道及調整深度學習網路。如下圖所述，神經網路的單純或基本實作使用 GPU 的方式可能不一致，而未完全發揮其潛能。當您最佳化您的前處理和資料載入時，您可以降低從 CPU 到 GPU 的瓶頸。您可以使用雜合 (當架構支援時)、調整批次大小和同步化呼叫，以調整神經網路本身。您在大多數架構中也可以使用多精確度 (float16 或 int8) 培訓，這可以大幅影響提高輸送量。

下列圖表顯示套用不同的最佳化時累積的效能提升。您的結果將取決於您處理的資料和您最佳化的網路。



範例 GPU 效能最佳化。圖表來源：[使用 MXNet Gluon 的效能技巧](#)

以下指南介紹了可與 DLAMI 配合使用的選項，並幫助您提高 GPU 性能。

主題

- [預處理](#)
- [培訓](#)

預處理

透過轉換或擴增的資料預先處理通常是 CPU 密集型程序，這可能成為整個管道中的瓶頸。架構有用於影像處理的內建運算子，但 DALI (Data Augmentation Library) 展現比架構的內建選項更好的效能。

- NVIDIA Data Augmentation Library (DALI) : DALI 將資料擴增卸載到 GPU。DLAMI 並未預先安裝，但您可以透過在 DLAMI 或其他 Amazon 彈性運算雲端執行個體上安裝或載入受支援的架構容器來存取它。如需詳細資訊，請參閱 NVIDIA 網站上的 [DALI 專案頁面](#)。如需範例使用案例和下載程式碼範例，請參閱 [SageMaker 預先處理訓練效能範例](#)。
- nvJPEG : 適用於 C 程式設計人員的 GPU 加速 JPEG 解碼器程式庫。它支援解碼單一映像或批次，以及深度學習中常見的後續轉換操作。nvJPEG 內建於 DALI，或者，您可以另外從 [NVIDIA 網站的 nvjpeg 頁面](#) 下載來使用。

對於 GPU 監控和最佳化，您可能有興趣看其他這些主題：

- [監控](#)
 - [使用以下方式監控 GPU CloudWatch](#)
- [最佳化](#)
 - [預處理](#)
 - [培訓](#)

培訓

透過混合精度培訓，您可以使用相同的記憶體數量來部署更大的網路，或相較於單一或雙精度網路而言，降低記憶體使用量，您將會看到運算效能提高。您也可享有較小和更快資料傳輸的好處，這是多節點分散式培訓的重要因素。若要利用混合精度培訓，您需要調整資料轉換和損耗縮放。以下指南說明如何對支援混合精度的架構這樣做。

- [NVIDIA 深度學習開發套件](#)-NVIDIA 網站上的文件，說明適用於 MXNet 的混合精確度實作，PyTorch 以及 TensorFlow

Tip

請務必到網站上查看您選擇的架構，並搜尋「混合精度」或 "fp16" 以取得最新的最佳化技術。以下是您可能覺得很有用的一些混合精度指南：

- [與 TensorFlow \(視頻 \) 混合精度培訓](#)-在 NVIDIA 博客網站。
- [使用 float16 搭配 MXNet 進行混合精度培訓](#) - MXNet 網站上的常見問答集文章。
- [NVIDIA 的頂點：輕鬆混合精度培訓的工具 PyTorch](#)-NVIDIA 網站上的博客文章。

對於 GPU 監控和最佳化，您可能有興趣看其他這些主題：

- [監控](#)
 - [使用以下方式監控 GPU CloudWatch](#)
- [最佳化](#)
 - [預處理](#)
 - [培訓](#)

具有 DL AWS AMI 的推論芯片

AWS Inferentia 是一種定制的機器學習芯片，其設計 AWS 可用於高性能推論預測。若要使用該晶片，請設定 Amazon 彈性運算雲端執行個體，並使用 AWS Neuron 軟體開發套件 (SDK) 呼叫推論晶片。為了為客戶提供最佳的推論體驗，神經元已內置於 AWS Deep Learning AMI (DLAMI) 中。

下列主題說明如何開始使用 DLAMI 的推論。

目錄

- [啟動含有神經元的 DLAMI 執行個體 AWS](#)
- [使用 DLAMI 與神經元 AWS](#)

啟動含有神經元的 DLAMI 執行個體 AWS

最新的 DLAMI 已準備好與 AWS 推論一起使用，並附帶神經元 API 包。AWS 若要啟動 DLAMI 執行個體，請參閱[啟動和設定 DLAMI](#)。取得 DLAMI 之後，請使用此處的步驟來確保您的 AWS 推論晶片和 AWS 神經元資源處於作用中狀態。

目錄

- [驗證您的執行個體](#)
- [識別 AWS 推論裝置](#)
- [檢視資源使用量](#)
- [使用神經元監視器 \(神經元監視器\)](#)
- [升級神經元軟件](#)

驗證您的執行個體

在使用您的執行個體之前，請確認執行個體是否已正確設定並設定 Neuron。

識別 AWS 推論裝置

若要識別執行個體上的 Inferentia 裝置數量，請使用下列指令：

```
neuron-ls
```

如果您的執行個體已連接 Inferentia 裝置，您的輸出會類似下列內容：

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

提供的輸出取自 Inf1.6xLarge 執行個體，並包含下列資料欄：

- 神經元裝置：指派給 NeuronDevice 配置多個運行時使用此 ID 以使用不同 NeuronDevices 的運行時。
- 神經元核心：目 NeuronCores 前在 NeuronDevice。
- 神經元記憶體：在 DRAM 記憶體的 NeuronDevice 量。
- 已連接的設備：其他 NeuronDevices 已連接到 NeuronDevice。
- PCI BDF：. 的 PCI 匯流排裝置功能 (BDF) 識別碼。NeuronDevice

檢視資源使用量

使用指 `neuron-top` 令檢視 vCPU 使用率、記憶體使用率、載入的模型和 Neuron 應用程式的有用資訊。NeuronCore 不 `neuron-top` 帶引數的啟動將顯示所有使用的機器學習應用程序的數據 NeuronCores。

```
neuron-top
```

當應用程式使用 4 時 NeuronCores，輸出應類似下列影像：


```

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

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

Loaded Models
[ - ] ND 0
[ - ] NC0
    -integ-tests/out-test7_resnet50_v2_fp16_b1_tpb1_tf
[ + ] NC1
[ + ] NC2
[ + ] NC3

Model ID          Host Memory          Device Memory
-----
10001             638.5KB              49.6MB
638.5KB           638.5KB              49.6MB
638.5KB           638.5KB              49.6MB
638.5KB           638.5KB              49.6MB

Neuron Apps
q: quit
arrows: move tree selection
enter: expand/collapse tree item
x: expand/collapse entire tree
a/d: previous/next tab
1-9: select tab

```

[如需監控和最佳化神經元推論應用程式的資源的詳細資訊，請參閱神經元工具。](#)

使用神經元監視器 (神經元監視器)

神經元監視器從系統上運行的神經元運行時收集指標，並以 JSON 格式將收集的數據流式傳輸到 stdout。這些測量結果會組織成您透過提供組態檔來設定的測量結果群組。有關神經元監視器的更多信息，請參閱[神經元監視器用戶指南](#)。

升級神經元軟件

[有關如何在 DLAMI 中更新神經元 SDK 軟件的信息，請參閱《AWS 神經元設置指南》。](#)

後續步驟

[使用 DLAMI 與神經元 AWS](#)

使用 DLAMI 與神經元 AWS

AWS Neuron SDK 的典型工作流程是在編譯伺服器上編譯先前訓練過的機器學習模型。在此之後，將成品分配到 Inf1 執行個體以便執行。AWS Deep Learning AMI (DLAMI) 已預先安裝在使用 Inferentia 的 Inf1 執行個體中編譯和執行推論所需的一切。

以下各節說明如何使用 DLAMI 與推論。

目錄

- [使用 TensorFlow-神經元和神經元編譯器 AWS](#)
- [使用 AWS 神經元服務 TensorFlow](#)
- [使用 MXNET 神經元和神經元編譯器 AWS](#)
- [使用 MXNet-Neuron 模型服務](#)
- [使用 PyTorch-神經元和神經元編譯器 AWS](#)

使用 TensorFlow-神經元和神經元編譯器 AWS

本教學課程說明如何使用 AWS 神經元編譯器來編譯 Keras ResNet -50 模型，並以格式匯出為已儲存的模型。SavedModel 此格式是與 TensorFlow 型的模型可互換格式。您也會學習如何使用範例輸入在 Inf1 執行個體上執行推論。

有關神經元 SDK 的更多信息，請參閱[AWS 神經元 SDK](#) 文檔。

目錄

- [必要條件](#)
- [啟動 Conda 環境](#)
- [Resnet50 編譯](#)
- [ResNet五十推論](#)

必要條件

在使用本教學課程之前，您應該已完成 [啟動含有神經元的 DLAMI 執行個體 AWS](#) 中的設置步驟。您還應該熟悉深度學習和 DLAMI 的使用。

啟動 Conda 環境

使用下列指令啟動 TensorFlow-神經元控制環境：

```
source activate aws_neuron_tensorflow_p36
```

若要結束目前的 conda 環境，請執行下列命令：

```
source deactivate
```

Resnet50 編譯

建立一個叫做 **tensorflow_compile_resnet50.py** 的 Python 指令碼，具有以下內容。這個 Python 腳本編譯凱拉斯 ResNet 50 模型，並將其導出為保存的模型。

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

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

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

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

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

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

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

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

使用下列命令編譯模型：

```
python tensorflow_compile_resnet50.py
```

編譯程序需要幾分鐘的時間。完成時，您的輸出應如以下所示：

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

編譯之後，儲存的模型會在 **ws_resnet50/resnet50_neuron.zip** 被壓縮。使用下列命令將模型解壓縮，並下載推論範例影像：

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

ResNet五十推論

建立一個叫做 **tensorflow_infer_resnet50.py** 的 Python 指令碼，具有以下內容。此指令碼使用先前編譯的推論模型，針對下載的模型執行推論。

```
import os
```

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

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

使用下列命令在模型上執行推論：

```
python tensorflow_infer_resnet50.py
```

您的輸出看起來應如以下所示：

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

後續步驟

[使用 AWS 神經元服務 TensorFlow](#)

使用 AWS 神經元服務 TensorFlow

本教學課程說明如何在匯出儲存的模型以與 Service 搭配 TensorFlow 使用之前，建構圖形並新增 AWS Neuron 編譯步驟。TensorFlow 服務是一種服務系統，可讓您跨網路擴展推論。神經元 TensorFlow 服務使用與正常 TensorFlow 服務相同的 API。唯一的區別是，必須針對 AWS Inferentia

編譯已儲存的模型，而入口點是名為的不同二進位檔案。tensorflow_model_server_neuron二進位檔可在 DLAMI 中找到/usr/local/bin/tensorflow_model_server_neuron並已預先安裝。

有關神經元 SDK 的更多信息，請參閱[AWS 神經元 SDK](#) 文檔。

目錄

- [必要條件](#)
- [啟動 Conda 環境](#)
- [編譯和匯出儲存的模型](#)
- [為儲存的模型提供服務](#)
- [向模型伺服器產生推論請求](#)

必要條件

在使用本教學課程之前，您應該已完成 [啟動含有神經元的 DLAMI 執行個體 AWS](#) 中的設置步驟。您還應該熟悉深度學習和 DLAMI 的使用。

啟動 Conda 環境

使用下列指令啟動 TensorFlow-神經元控制環境：

```
source activate aws_neuron_tensorflow_p36
```

如果您需要退出目前的 conda 環境，請執行：

```
source deactivate
```

編譯和匯出儲存的模型

創建一個使用以下內容調tensorflow-model-server-compile.py用的 Python 腳本。這個腳本構造一個圖形，並使用神經元編譯它。然後將編譯後的圖形匯出為儲存的模型。

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

```
import os

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

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

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

使用下列命令編譯模型：

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

您的輸出看起來應如以下所示：

```
...
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
```

為儲存的模型提供服務

一旦模型已編譯過，您可以使用以下命令，以 `tensorflow_model_server_neuron` 二進位檔案為儲存的模型提供服務：

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

您的輸出看起來應該如下所示。伺服器會在推論裝置的 DRAM 中暫存編譯的模型，以準備進行推論。

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

向模型伺服器產生推論請求

建立一個叫做 `tensorflow-model-server-infer.py` 的 Python 指令碼，具有以下內容。此指令碼透過 gRPC (為一服務框架) 執行推斷。

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

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



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

使用 gRPC，以下列命令在模型上執行推論：

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

您的輸出看起來應如以下所示：

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

使用 MXNET 神經元和神經元編譯器 AWS

MXNET 神經元編譯 API 提供了一種方法來編譯可在推論裝置上執行的模型圖形。AWS

在此範例中，您可以使用 API 編譯 ResNet -50 模型，並使用它來執行推論。

有關神經元 SDK 的更多信息，請參閱[AWS 神經元 SDK](#) 文檔。

目錄

- [必要條件](#)
- [啟動 Conda 環境](#)
- [Resnet50 編譯](#)
- [ResNet五十推論](#)

必要條件

在使用本教學課程之前，您應該已完成 [啟動含有神經元的 DLAMI 執行個體 AWS](#) 中的設置步驟。您還應該熟悉深度學習和 DLAMI 的使用。

啟動 Conda 環境

使用以下命令啟動 MXNet-Neuron conda 環境：

```
source activate aws_neuron_mxnet_p36
```

若要退出目前的 conda 環境，請執行：

```
source deactivate
```

Resnet50 編譯

建立一個叫做 **mxnet_compile_resnet50.py** 的 Python 指令碼，具有以下內容。此指令碼使用 MXNET 神經元編譯 Python API 來編譯 -50 模型。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)
```

使用下列命令編譯模型：

```
python mxnet_compile_resnet50.py
```

編譯將需要幾分鐘的時間。編譯完成後，下列檔案將位於您目前的目錄中：

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

ResNet五十推論

建立一個叫做 `mxnet_infer_resnet50.py` 的 Python 指令碼，具有以下內容。此指令碼會下載範例影像，並使用它來執行具有已編譯模型的推論。

```
import mxnet as mx
import numpy as np

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

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

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

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

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

exe.forward(data=img)
prob = exe.outputs[0].asnumpy()
```

```
# print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' %(prob[i], labels[i]))
```

使用以下命令，以編譯模型執行推斷：

```
python mxnet_infer_resnet50.py
```

您的輸出看起來應如以下所示：

```
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
```

後續步驟

[使用 MXNet-Neuron 模型服務](#)

使用 MXNet-Neuron 模型服務

在本教學課程中，您將學習如何使用預先培訓的 MXNet 模型來執行多模型伺服器 (MMS) 的即時影像分類。MMS 是一種靈活的 easy-to-use 工具，可提供使用任何機器學習或深度學習架構訓練的深度學習模型。本教學課程包含使用 AWS 神經元的編譯步驟，以及使用 MXNet 實作的 MMS。

有關神經元 SDK 的更多信息，請參閱[AWS 神經元 SDK](#) 文檔。

目錄

- [必要條件](#)
- [啟動 Conda 環境](#)
- [下載範例程式碼](#)
- [編譯模型](#)
- [執行推論](#)

必要條件

在使用本教學課程之前，您應該已完成 [啟動含有神經元的 DLAMI 執行個體 AWS](#) 中的設置步驟。您還應該熟悉深度學習和 DLAMI 的使用。

啟動 Conda 環境

使用以下命令啟動 MXNet-Neuron conda 環境：

```
source activate aws_neuron_mxnet_p36
```

若要退出目前的 conda 環境，請執行：

```
source deactivate
```

下載範例程式碼

若要執行此範例，請使用下列命令下載範例程式碼：

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

編譯模型

建立一個叫做 `multi-model-server-compile.py` 的 Python 指令碼，具有以下內容。該腳本將 ResNet 50 模型編譯為推論設備目標。

```
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)
```

若要編譯模型，請使用下列命令：

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

您的輸出看起來應如以下所示：

```
...
[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!
```

建立一個叫做 `signature.json` 的檔案，具有以下內容，以設定輸入名稱和形狀：

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

您可以使用下列命令來下載 `synset.txt` 檔案。此檔案是 ImageNet 預測類別的名稱清單。

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

按照 `model_server_template` 資料夾中的範本建立自訂服務類別。使用下列命令將範本複製到目前的工作目錄：

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

編輯 `mxnet_model_service.py` 模組，將 `mx.cpu()` 內容取代為 `mx.neuron()` 內容，如下所示。您還需要將 `model_input` 所使用、不必要的資料副本註釋掉，因為 MXNet-Neuron 不支援 `NDArray` 和 `Gluon API`。

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

使用下列指令，以模型歸檔程式封裝模型：

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

執行推論

啟動多模型伺服器，並使用下列命令載入使用 RESTful API 的模型。請確認 `neuron-rtd` 是以預設設定執行。

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

以下列命令使用範例影像執行推論：

```
curl -O https://raw.githubusercontent.com/awslabs/multi-model-server/master/docs/images/kitten_small.jpg
```

```
curl -X POST http://127.0.0.1:8080/predictions/resnet-50_compiled -T kitten_small.jpg
```

您的輸出看起來應如以下所示：

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

若要在測試之後進行清理，請透過 RESTful API 發出 delete 命令，並使用以下命令停止模型伺服器：

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

multi-model-server --stop
```

您應該會看到下列輸出：

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


使用 PyTorch-神經元和神經元編譯器 AWS

PyTorch-Neuron 編譯 API 提供了一種方法來編譯您可以在 AWS 推論設備上運行的模型圖。

經過訓練的模型必須編譯至 Inferentia 目標，才能部署到 Inf1 執行個體上。以下教學課程編譯了 Torchvision ResNet 50 模型，並將其匯出為儲存的模組。TorchScript 接著，即可使用此模型執行推論。

為了方便起見，本教學課程使用 Inf1 執行個體進行編譯和推論。實際上，您可以使用 c5 執行個體系列等其他執行個體類型來編譯模型。接著，您必須將已編譯的模型部署到 Inf1 推論伺服器。如需詳細資訊，請參閱[AWS 神經元 PyTorch SDK 文件](#)。

目錄

- [必要條件](#)
- [啟動 Conda 環境](#)
- [Resnet50 編譯](#)
- [ResNet五十推論](#)

必要條件

在使用本教學課程之前，您應該已完成 [啟動含有神經元的 DLAMI 執行個體 AWS](#) 中的設置步驟。您還應該熟悉深度學習和 DLAMI 的使用。

啟動 Conda 環境

使用下列指令啟動 PyTorch-神經元控制環境：

```
source activate aws_neuron_pytorch_p36
```

若要退出目前的 conda 環境，請執行：

```
source deactivate
```

Resnet50 編譯

建立一個叫做 `pytorch_trace_resnet50.py` 的 Python 指令碼，具有以下內容。這個腳本使用 PyTorch-神經元編譯 Python API 來編譯一個 ResNet -50 模型。

Note

在編譯火炬視覺模型時，您應該注意這些版本和火炬軟件包之間存在依賴關係。這些依賴規則可以通過 pip 進行管理。火炬視野 ==0.6.1 與火炬 ==1.5.1 版本相符，而火炬視野 ==0.8.2 與火炬 ==1.7.1 版本相符。

```
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")
```

執行編譯指令碼。

```
python pytorch_trace_resnet50.py
```

編譯將需要幾分鐘的時間。編譯完成後，編譯後的模型將另存為 `resnet50_neuron.pt` 本地目錄中。

ResNet五十推論

建立一個叫做 `pytorch_infer_resnet50.py` 的 Python 指令碼，具有以下內容。此指令碼會下載範例影像，並使用它來執行具有已編譯模型的推論。

```
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/aws-labs/mxnet-model-server/
master/docs/images/kitten_small.jpg",
                    "./torch_neuron_test/images/kitten_small.jpg")

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

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

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

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

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

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

```
## Predict
results = model_neuron( image )

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

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

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

使用以下命令，以編譯模型執行推斷：

```
python pytorch_infer_resnet50.py
```

您的輸出看起來應如以下所示：

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

重力子 DLAMI

AWS Graviton GPU DLAMI 專為深度學習工作負載提供高效能與成本效益而設計。具體而言，G5G 執行個體類型採用以 ARM 為基礎的 [AWS Graviton2 處理器](#)，該處理器是從頭開始構建的，AWS 並針對客戶在雲端中執行工作負載的方式進行了優化。AWS 引力子 GPU DlamI 已預先配置為泊塢視窗、NVIDIA 泊塢視窗、NVIDIA 驅動程式、CUDA、CuDNN N、非 CCL 和 TensorRT，以及熱門的機器學習架構，例如和。TensorFlow PyTorch

使用 G5G 執行個體類型，與具有 GPU 加速的 x86 型執行個體相比，您可以利用 Graviton2 的價格和效能優勢來部署 GPU 加速深度學習模型，成本大幅降低。

選擇一個重力 DLAMI

使用您選擇的重力子 DLAMI 啟動 [G5G 執行個體](#)。

如需啟動 DLAMI 的 step-by-step 指示，請參閱[啟動和設定 DLAMI](#)。

如需最新引力子 DLAMI 的清單，請參閱 DLAMI 的[發行說明](#)。

開始使用

下列主題向您展示如何開始使用重力子 DLAMI。

目錄

- [使用引力子 GPU 直接封裝置](#)
- [使用引力子 GPU 直接封裝置 TensorFlow](#)
- [使用引力子 GPU 直接封裝置 PyTorch](#)

使用引力子 GPU 直接封裝置

可與 Arm 處理器為基礎的 AWS Deep Learning AMI 重力子 GPU 搭配使用。Graviton GPU DLAMI 隨附 GPU 驅動程式和加速程式庫的基礎平台，可部署您自訂的深度學習環境。泊塢視窗和 NVIDIA 泊塢視窗已在重力磁碟機 DLAMI 上預先設定，讓您部署容器化的應用程式。如需有關引力子 GPU DLAMI 的其他詳細資訊，請查看[版本說明](#)。

目錄

- [檢查 GPU 狀態](#)
- [檢查 CUDA 版本](#)
- [驗證碼頭工人](#)
- [TensorRT](#)
- [執行 CUDA 範例](#)

檢查 GPU 狀態

使用 [NVIDIA 系統管理介面](#) 來檢查重力彈 GPU 的狀態。

```
nvidia-smi
```

nvidia-smi 命令的輸出應該類似於以下內容：

```
+-----+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4    |
+-----+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf   Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                               |                  |           MIG M.     |
+-----+-----+-----+-----+
```

```

=====+=====+=====
|  0  NVIDIA T4G          On | 00000000:00:1F.0 Off |          0 |
| N/A  32C   P8          8W / 70W |          0MiB / 15109MiB |          0%   Default |
|                                     |                                     |          N/A |
+-----+-----+-----+

+-----+
| Processes: |
| GPU  GI   CI          PID  Type   Process name          GPU Memory |
|      ID  ID                                     Usage          |
+-----+
| No running processes found |
+-----+

```

檢查 CUDA 版本

執行下列命令以檢查您的 CUDA 版本：

```
/usr/local/cuda/bin/nvcc --version | grep Cuda
```

您的輸出應該類似以下內容：

```
nvcc: NVIDIA (R) Cuda compiler driver
Cuda compilation tools, release 11.4, V11.4.120
```

驗證碼頭工人

執行 CUDA 容器，[DockerHub](https://hub.docker.com/r/nvidia/cuda)以驗證引力士 GPU 上的碼頭工人功能：

```
sudo docker run --platform=linux/arm64 --rm \
  --gpus all nvidia/cuda:11.4.2-base-ubuntu20.04 nvidia-smi
```

您的輸出應該類似以下內容：

```

+-----+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4    |
+-----+-----+-----+
| GPU  Name            Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                     |                  |           MIG M.     |
+-----+-----+-----+
|  0  NVIDIA T4G          On | 00000000:00:1F.0 Off |          0 |

```

```

| N/A   33C   P8     9W / 70W |           0MiB / 15109MiB |           0%           Default |
|                                           |                           |                           |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
| Processes:                               |                           |                           |
| GPU   GI   CI           PID   Type   Process name                      GPU Memory |
|       ID   ID                                   Usage                          |
|=====|
| No running processes found              |                           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

TensorRT

使用下列命令來存取 TensorRT 指令列工具：

```
trtexec
```

您的輸出應該類似以下內容：

```

&&&& RUNNING TensorRT.trtexec [TensorRT v8200] # trtexec
...
&&&& PASSED TensorRT.trtexec [TensorRT v8200] # trtexec

```

有提供按需安裝的 TensorRT Python 輪。您可以在下列檔案位置找到這些操控盤：

```

/usr/local/tensorrt/graphsurgeon/
### graphsurgeon-0.4.5-py2.py3-none-any.whl

/usr/local/tensorrt/onnx_graphsurgeon/
### onnx_graphsurgeon-0.3.12-py2.py3-none-any.whl

/usr/local/tensorrt/python/
### tensorrt-8.2.0.6-cp36-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp37-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp38-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp39-none-linux_aarch64.whl

/usr/local/tensorrt/uff/
### uff-0.6.9-py2.py3-none-any.whl

```

如需其他詳細資訊，請參閱 [NVIDIA TensorRT 說明文件](#)。

執行 CUDA 範例

引力子 GPU DLAMI 提供預先編譯的 CUDA 範例，協助您驗證不同的 CUDA 功能。

```
ls /usr/local/cuda/compiled_samples
```

例如，使用下列命令執行vectorAdd範例：

```
/usr/local/cuda/compiled_samples/vectorAdd
```

您的輸出應該類似以下內容：

```
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

執行transpose範例：

```
/usr/local/cuda/compiled_samples/transpose
```

您的輸出應該類似以下內容：

```
Transpose Starting...

GPU Device 0: "Turing" with compute capability 7.5

> Device 0: "NVIDIA T4G"
> SM Capability 7.5 detected:
> [NVIDIA T4G] has 40 MP(s) x 64 (Cores/MP) = 2560 (Cores)
> Compute performance scaling factor = 1.00

Matrix size: 1024x1024 (64x64 tiles), tile size: 16x16, block size: 16x16

transpose simple copy          , Throughput = 185.1781 GB/s, Time = 0.04219 ms, Size =
  1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose shared memory copy, Throughput = 163.8616 GB/s, Time = 0.04768 ms, Size =
  1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
```



```
transpose naive           , Throughput = 98.2805 GB/s, Time = 0.07949 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose coalesced      , Throughput = 127.6759 GB/s, Time = 0.06119 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose optimized      , Throughput = 156.2960 GB/s, Time = 0.04999 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose coarse-grained , Throughput = 155.9157 GB/s, Time = 0.05011 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose fine-grained   , Throughput = 158.4177 GB/s, Time = 0.04932 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose diagonal      , Throughput = 133.4277 GB/s, Time = 0.05855 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
Test passed
```

接下來

[使用引力子 GPU 直接封裝置 TensorFlow](#)

使用引力子 GPU 直接封裝置 TensorFlow

可與 Arm 處理器為基礎的重力子 GPU 搭配使用，並 AWS Deep Learning AMI 針對 TensorFlow 引力子 GPU TensorFlow DLAMI 包含預先設定的 Python 環境，並提供深度學習推論使用案例的「[TensorFlow 服務](#)」。如需有關引力子 GPU TensorFlow DLAMI 的其他詳細資訊，請查看[版本說明](#)。

目錄

- [驗證 TensorFlow 服務可用性](#)
- [驗證 TensorFlow 和 TensorFlow 提供 API 可用性](#)
- [使 TensorFlow 用服務執行範例推論](#)

驗證 TensorFlow 服務可用性

執行下列命令以驗證 Serve 的可用性和版 TensorFlow 本：

```
tensorflow_model_server --version
```

您的輸出應該類似以下內容：

```
TensorFlow ModelServer: 0.0.0+dev.sha.3e05381e
TensorFlow Library: 2.8.0
```

驗證 TensorFlow 和 TensorFlow 提供 API 可用性

執行下列命令以驗證 TensorFlow 和 TensorFlow 服務 API 的可用性：

```
python3 -c "import tensorflow, tensorflow_serving"
```

如果命令成功，則沒有輸出。

使 TensorFlow 用服務執行範例推論

使用下列命令下載預先訓練的 ResNet 50 模型，並使 TensorFlow 用 Serve 執行推論：

```
# Clone the TensorFlow Serving repository
git clone https://github.com/tensorflow/serving

# Download pre-trained ResNet50 model
mkdir -p ${HOME}/resnet/1 && cd ${HOME}/resnet/1
wget https://tfhub.dev/tensorflow/resnet_50/classification/1?tf-hub-format=compressed -O resnet_50_classification_1.tar.gz
tar -xzvf resnet_50_classification_1.tar.gz && rm resnet_50_classification_1.tar.gz

# Start TensorFlow Serving
cd $HOME
tensorflow_model_server \
  --rest_api_port=8501 \
  --model_name="resnet" \
  --model_base_path="${HOME}/resnet" &
```

您的輸出應該類似以下內容：

```
2021-11-10 06:18:51.028341: I tensorflow_serving/model_servers/server_core.cc:486]
  Finished adding/updating models
2021-11-10 06:18:51.028420: I tensorflow_serving/model_servers/server.cc:133] Using
  InsecureServerCredentials
2021-11-10 06:18:51.028460: I tensorflow_serving/model_servers/server.cc:383] Profiler
  service is enabled
2021-11-10 06:18:51.028889: I tensorflow_serving/model_servers/server.cc:409] Running
  gRPC ModelServer at 0.0.0.0:8500 ...
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
2021-11-10 06:18:51.030985: I tensorflow_serving/model_servers/server.cc:430] Exporting
  HTTP/REST API at:localhost:8501 ...
```

使用 Ser TensorFlow vice resnet_client [範例](#)來執行推論：

```
python3 serving/tensorflow_serving/example/resnet_client.py
```

您的輸出應該類似以下內容：

```
2021-11-10 06:18:59.335327: I external/org_tensorflow/tensorflow/stream_executor/cuda/cuda_dnn.cc:368] Loaded cuDNN version 8204
2021-11-10 06:18:59.956156: I external/org_tensorflow/tensorflow/core/platform/default/subprocess.cc:304] Start cannot spawn child process
Prediction class: 285, avg latency: 111.4673 ms
```

使用以下命令停止 TensorFlow 服務：

```
kill $(pidof tensorflow_model_server)
```

接下來

[使用引力子 GPU 直接封裝置 PyTorch](#)

使用引力子 GPU 直接封裝置 PyTorch

可與 Arm 處理器為基礎的重力子 GPU 搭配使用，並 AWS Deep Learning AMI 針對 PyTorch 引力子 GPU PyTorch DLAMI 包含預先設定 [PyTorch](#)、[TorchVision](#) 以及深度學習訓練和推論使用 [TorchServe](#) 例的 Python 環境。如需有關引力子 GPU PyTorch DLAMI 的其他詳細資訊，請查看 [版本說明](#)。

目錄

- [驗證 PyTorch Python 境](#)
- [運行訓練示例 PyTorch](#)
- [執行推論範例 PyTorch](#)

驗證 PyTorch Python 境

Connect 到您的 G5G 實例，並使用以下命令激活基本 Conda 環境：

```
source activate base
```

您的指令提示字元應指出您正在基礎 Conda 環境中工作，其中包含 PyTorch TorchVision、和其他程式庫。

```
(base) $
```

驗證 PyTorch 環境的預設刀具路徑：

```
(base) $ which python
/opt/conda/bin/python
```

```
(base) $ which pip
/opt/conda/bin/pip
```

```
(base) $ which conda
/opt/conda/bin/conda
```

```
(base) $ which mamba
/opt/conda/bin/mamba
```

驗證火炬和可 TorchVersion 用，檢查其版本，並測試基本功能：

```
(base) $ python
Python 3.8.12 | packaged by conda-forge | (default, Oct 12 2021, 23:06:28)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch, torchvision
>>> torch.__version__
'1.10.0'
>>> torchvision.__version__
'0.11.1'
>>> 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)
```

運行訓練示例 PyTorch

執行 MNIST 訓練工作範例：

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

您的輸出應該類似以下內容：

```
...
```

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

執行推論範例 PyTorch

使用下列指令來下載預先訓練的 densenet161 模型，並使用下列方式執行推論： TorchServe

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

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

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

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

# Wait for the model server to start
sleep 30

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

您的輸出應該類似以下內容：

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

使用下列命令取消註冊 densenet161 模型並停止伺服器：

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

您的輸出應該類似以下內容：

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

哈瓦那 DLAMI

具有 Habana 加速器的執行個體旨在為深度學習模型訓練工作負載提供高效能和成本效益。具體來說，DL1 執行個體類型使用來自英特爾公司哈瓦那實驗室的哈瓦那高迪加速器。具有 Habana 加速器的執行個體配置了 Habana SynapseAI 軟體，並預先與流行的機器學習框架（例如和）整合。TensorFlow PyTorch

下列主題說明如何開始使用 Habana 高迪硬體搭配 DLAMI。

目錄

- [啟動哈瓦那 DLAMI](#)

啟動哈瓦那 DLAMI

最新的 DLAMI 已準備好與哈瓦那高迪加速器一起使用。使用以下步驟啟動您的哈瓦那 DLAMI，並確保您的 Python 和框架特定資源處於活動狀態。如需其他安裝資源，請參閱 [Habana Gaudi 安裝和安裝儲存庫](#)。

目錄

- [選擇一個哈瓦那 DLAMI](#)
- [啟動 Python 環境](#)
- [匯入 Machine Learning 架構](#)

選擇一個哈瓦那 DLAMI

使用您選擇的哈瓦那 [DLAMI 啟動 DL1 執行個體](#)。

如需啟動 DLAMI 的 step-by-step 指示，請參閱[啟動和設定 DLAMI](#)。

如需最新的哈瓦那 DLAMI 清單，請參閱 DLAMI 的[發行說明](#)。

啟動 Python 環境

Connect 至您的 DL1 執行個體，並為您的哈瓦那 DLAMI 啟動建議的 Python 環境。若要檢查您建議的 Python 環境，請在[版本](#)說明中選取您的 DLAMI。

匯入 Machine Learning 架構

具有 Habana 加速器的執行個體已預先與熱門機器學習架構 (例如和) 整合。TensorFlow PyTorch 匯入您選擇的機器學習架構。

進口 TensorFlow

要在哈瓦那 DLAMI TensorFlow 上使用，請導航到您激活並導入的 Python 環境的文件夾。

TensorFlow

```
/usr/bin/$PYTHON_VERSION
import tensorflow
tensorflow.__version__
```

[若要檢查與您的 Habana DLAMI 相容的 TensorFlow 版本，請在發行說明中選取您的 DLAMI。](#)

進口 PyTorch

要在哈瓦那 DLAMI PyTorch 上使用，請導航到您激活的 Python 環境的文件夾並導入適當的版本。

PyTorch

```
/usr/bin/$PYTHON_VERSION
import torch
```

```
torch.__version__
```

[若要檢查與您的 Habana DLAMI 相容的 PyTorch 版本，請在發行說明中選取您的 DLAMI。](#)

[有關如何在 Habana DLAMI 中運行和訓練機器學習模型的 TensorFlow 更多信息，請參閱 Habana 模型參考存儲庫。PyTorch GitHub 如需使用 Habana DLAMI 的其他資源，請瀏覽\[哈瓦那高迪文件\]\(#\)。](#)

Inference

本節提供如何使用 DLAMI 架構和工具執行推論的教學課程。

如需使用 Elastic Inference 的教學課程，請參閱[使用 Amazon Elastic Inference](#)

推論與架構

- [使用阿帕奇 MXNet \(孵化 \) 進行 ONNX 模型的推論](#)
- [使用阿帕奇 MXNet \(孵化 \) 進行 50 模型的推論 ResNet](#)
- [使用 CNTK 搭配 ONNX 模型進行推論](#)

推論工具

- [Apache MXNet 專用模型伺服器 \(MMS\)](#)
- [TensorFlow 服務](#)

使用阿帕奇 MXNet (孵化) 進行 ONNX 模型的推論

如何使用 ONNX 模型進行圖像推論與阿帕奇 MXNet (孵化)

1. • (Python 3 的選項) -激活 Python 3 阿帕奇 MXNet (孵化) 環境：

```
$ source activate mxnet_p36
```

- (Python 2 的選項) -激活 Python 2 阿帕奇 MXNet (孵化) 環境：

```
$ source activate mxnet_p27
```

2. 其餘步驟假設您使用的是 mxnet_p36 環境。
3. 下載哈士奇的圖片。


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

4. 下載可與此模型搭配運作的類別清單。

```
$ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsidx_to_human.pkl
```

5. 下載 ONNX 格式的預先訓練 VGG 16 模型。

```
$ wget -O vgg16.onnx https://github.com/onnx/models/raw/master/vision/classification/vgg/model/vgg16-7.onnx
```

6. 使用您慣用的文字編輯器來建立具有下列內容的指令碼。此指令碼將使用哈士奇的影像、從預先訓練的模型中取得預測結果，然後在類別檔案中查閱此項目，傳回映像分類結果。

```
import mxnet as mx
import mxnet.contrib.onnx as onnx_mxnet
import numpy as np
from collections import namedtuple
from PIL import Image
import pickle

# Preprocess the image
img = Image.open("Siberian_Husky_bi-eyed_Flickr.jpg")
img = img.resize((224,224))
rgb_img = np.asarray(img, dtype=np.float32) - 128
bgr_img = rgb_img[..., [2,1,0]]
img_data = np.ascontiguousarray(np.rollaxis(bgr_img,2))
img_data = img_data[np.newaxis, :, :, :].astype(np.float32)

# Define the model's input
data_names = ['data']
Batch = namedtuple('Batch', data_names)

# Set the context to cpu or gpu
ctx = mx.cpu()

# Load the model
sym, arg, aux = onnx_mxnet.import_model("vgg16.onnx")
mod = mx.mod.Module(symbol=sym, data_names=data_names, context=ctx,
label_names=None)
```

```

mod.bind(for_training=False, data_shapes=[(data_names[0],img_data.shape)],
        label_shapes=None)
mod.set_params(arg_params=arg, aux_params=aux, allow_missing=True,
              allow_extra=True)

# Run inference on the image
mod.forward(Batch([mx.nd.array(img_data)]))
predictions = mod.get_outputs()[0].asnumpy()
top_class = np.argmax(predictions)
print(top_class)
labels_dict = pickle.load(open("imagenet1000_clsids_to_human.pkl", "rb"))
print(labels_dict[top_class])

```

7. 然後執行指令碼，您應該會看到如下所示的結果：

```

248
Eskimo dog, husky

```

使用阿帕奇 MXNet (孵化) 進行 50 模型的推論 ResNet

如何使用預先訓練的 Apache MXNet (孵化) 模型搭配使用符號 API，以便透過 MXNet 進行影像推論

1. • (Python 3 的選項) - 激活 Python 3 阿帕奇 MXNet (孵化) 環境：

```
$ source activate mxnet_p36
```

• (Python 2 的選項) - 激活 Python 2 阿帕奇 MXNet (孵化) 環境：

```
$ source activate mxnet_p27
```

2. 其餘步驟假設您使用的是 mxnet_p36 環境。

3. 使用您慣用的文字編輯器來建立具有下列內容的指令碼。該腳本將下載 ResNet -50 模型文件 (resnet-50-0000.params 和 resnet-50-符號l.json) 和標籤列表 (synset.txt)，下載一個貓圖像以從預先訓練的模型中獲取預測結果，然後在標籤列表中查找結果，然後在標籤列表中查找結果，返回預測結果。

```

import mxnet as mx
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')]

ctx = mx.cpu()

with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]

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

fname = mx.test_utils.download('https://github.com/dmlc/web-data/blob/master/mxnet/
doc/tutorials/python/predict_image/cat.jpg?raw=true')
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')
args['data'] = img

softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax

exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')

exe.forward()
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]))
```

4. 然後執行指令碼，您應該會看到如下所示的結果：

```
probability=0.418679, class=n02119789 kit fox, Vulpes macrotis
probability=0.293495, class=n02119022 red fox, Vulpes vulpes
probability=0.029321, class=n02120505 grey fox, gray fox, Urocyon cinereoargenteus
probability=0.026230, class=n02124075 Egyptian cat
probability=0.022557, class=n02085620 Chihuahua
```

使用 CNTK 搭配 ONNX 模型進行推論

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

Note

本教學課程所使用的 VGG-16 模型會耗用大量記憶體。選取 AWS Deep Learning AMI 執行個體時，您可能需要具有 30 GB 以上 RAM 的執行個體。

如何使用 ONNX 模型來以 CNTK 進行推論

- (適用於 Python 3 的選項) - 啟用 Python 3 CNTK 環境：

```
$ source activate cntk_p36
```

- (適用於 Python 2 的選項) - 啟用 Python 2 CNTK 環境：

```
$ source activate cntk_p27
```

- 其餘步驟假設您使用的是 cntk_p36 環境。
- 使用文字編輯器建立新的檔案，並使用指令碼中的以下程式在 CNTK 中開啟 ONNX 格式檔案。

```
import cntk as C
# Import the Chainer model into CNTK via the CNTK import API
z = C.Function.load("vgg16.onnx", device=C.device.cpu(), format=C.ModelFormat.ONNX)
print("Loaded vgg16.onnx!")
```

執行此指令碼後，CNTK 將會載入模型。

- 您也可以嘗試使用 CNTK 進行推論。首先，下載哈士奇的圖片。

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

- 接著，下載搭配此模型運作的類別清單。

```
$ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsidx_to_human.pkl
```

- 編輯之前建立的指令碼，使其具有以下內容。這個新的版本將使用哈士奇的影像、取得預測結果，然後在類別檔案中尋找此項目，傳回預測結果。

```
import cntk as C
import numpy as np
from PIL import Image
from IPython.core.display import display
import pickle

# Import the model into CNTK via the CNTK import API
z = C.Function.load("vgg16.onnx", device=C.device.cpu(), format=C.ModelFormat.ONNX)
print("Loaded vgg16.onnx!")
img = Image.open("Siberian_Husky_bi-eyed_Flickr.jpg")
img = img.resize((224,224))
rgb_img = np.asarray(img, dtype=np.float32) - 128
bgr_img = rgb_img[..., [2,1,0]]
img_data = np.ascontiguousarray(np.rollaxis(bgr_img,2))
predictions = np.squeeze(z.eval({z.arguments[0]:[img_data]}))
top_class = np.argmax(predictions)
print(top_class)
labels_dict = pickle.load(open("imagenet1000_clsidx_to_human.pkl", "rb"))
print(labels_dict[top_class])
```

- 然後執行指令碼，您應該會看到如下所示的結果：

```
248
Eskimo dog, husky
```

使用架構與 ONNX

搭配 Conda 的深度學習 AMI 現在支援某些架構的[開放神經網路交換 \(ONNX\)](#) 模型。選擇下列其中一個主題，了解如何在 Conda 的深度學習 AMI 上使用 ONNX。

如果您要在 DLAMI 上使用現有的 ONNX 模型，請參閱。[使用阿帕奇 MXNet \(孵化 \) 進行 ONNX 模型的推論](#)

關於 ONNX

[開放神經網路交換 \(ONNX\)](#) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

使用 Conda 的深度學習 AMI 目前在以下教程集中突出了一些 ONNX 功能。

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)

您也可以參閱 ONNX 專案文件和教學課程：

- [上海 ONNX 專案 GitHub](#)
- [ONNX 教學課程](#)

Apache MXNet 到 ONNX 到 CNTK 教學課程

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

ONNX 概觀

[開放神經網路交換 \(ONNX\)](#) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

本教學課程說明如何搭配 ONNX 搭配 Conda 使用深度學習 AMI。遵循下列步驟，您可以訓練模型或從一個架構載入預先訓練的模型、將此模型匯出到 ONNX，然後將此模型匯入到另一個架構。

ONNX 先決條件

若要使用此 ONNX 教學課程，您必須能夠存取具有 Conda 版本 12 或更新版本的深度學習 AMI。如需如何透過 Conda 開始使用深度學習 AMI 的詳細資訊，請參閱[使用康達深度學習 AMI](#)。

Important

這些範例使用的函數可能需要最多 8 GB 的記憶體 (或更多)。請務必選擇具有足夠記憶體的執行個體類型。

使用 Conda 使用深度學習 AMI 啟動終端工作階段，開始下列教學課程。

將 Apache MXNet (incubating) 模型轉換到 ONNX，然後將模型載入 CNTK

如何從 Apache MXNet (incubating) 匯出模型

您可以使用 Conda 在深度學習 AMI 上將最新的 MXNet 組建安裝到 MXNet Conda 環境中的一個或兩個環境中。

- (適用於 Python 3 的選項) - 啟用 Python 3 MXNet 環境：

```
$ source activate mxnet_p36
```

- (適用於 Python 2 的選項) - 啟用 Python 2 MXNet 環境：

```
$ source activate mxnet_p27
```

2. 其餘步驟假設您使用的是 mxnet_p36 環境。

3. 下載模型檔案。

```
curl -O https://s3.amazonaws.com/onnx-mxnet/model-zoo/vgg16/vgg16-symbol.json
curl -O https://s3.amazonaws.com/onnx-mxnet/model-zoo/vgg16/vgg16-00000.params
```

4. 若要從 MXNet 匯出模型檔案到 ONNX 格式，請使用文字編輯器建立新的檔案，並在指令碼中使用以下程式。

```
import numpy as np
import mxnet as mx
from mxnet.contrib import onnx as onnx_mxnet
converted_onnx_filename='vgg16.onnx'
```

```
# Export MXNet model to ONNX format via MXNet's export_model API
converted_onnx_filename=onnx_mxnet.export_model('vgg16-symbol.json',
        'vgg16-0000.params', [(1,3,224,224)], np.float32, converted_onnx_filename)

# Check that the newly created model is valid and meets ONNX specification.
import onnx
model_proto = onnx.load(converted_onnx_filename)
onnx.checker.check_model(model_proto)
```

您可能會看到一些警告訊息，但目前可以放心地忽略。執行此指令碼後，您將在相同的目錄中看到新建立的 .onnx 檔案。

5. 既然您具有 ONNX 檔案，您就可以使用以下範例，嘗試搭配此檔案執行推論：

- [使用 CNTK 搭配 ONNX 模型進行推論](#)

ONNX 教學課程

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

Chainer 到 ONNX 到 CNTK 教學課程

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

ONNX 概觀

[開放神經網路交換](#) (ONNX) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

本教學課程說明如何搭配 ONNX 搭配 Conda 使用深度學習 AMI。遵循下列步驟，您可以訓練模型或從一個架構載入預先訓練的模型、將此模型匯出到 ONNX，然後將此模型匯入到另一個架構。

ONNX 先決條件

若要使用此 ONNX 教學課程，您必須能夠存取具有 Conda 版本 12 或更新版本的深度學習 AMI。如需如何透過 Conda 開始使用深度學習 AMI 的詳細資訊，請參閱[使用康達深度學習 AMI](#)。

Important

這些範例使用的函數可能需要最多 8 GB 的記憶體 (或更多)。請務必選擇具有足夠記憶體的執行個體類型。

使用 Conda 使用深度學習 AMI 啟動終端工作階段，開始下列教學課程。

將 Chainer 轉換到 ONNX，然後將模型載入 CNTK

首先，啟用 Chainer 環境：

```
$ source activate chainer_p36
```

使用文字編輯器建立新的檔案，並使用指令碼中的以下程式從 Chainer model zoo 擷取模型，然後將其匯出到 ONNX 格式。

```
import numpy as np
import chainer
import chainercv.links as L
import onnx_chainer

# Fetch a vgg16 model
model = L.VGG16(pretrained_model='imagenet')

# Prepare an input tensor
x = np.random.rand(1, 3, 224, 224).astype(np.float32) * 255

# Run the model on the data
with chainer.using_config('train', False):
    chainer_out = model(x).array
```

```
# Export the model to a .onnx file
out = onnx_chainer.export(model, x, filename='vgg16.onnx')

# Check that the newly created model is valid and meets ONNX specification.
import onnx
model_proto = onnx.load("vgg16.onnx")
onnx.checker.check_model(model_proto)
```

執行此指令碼後，您將在相同的目錄中看到新建立的 .onnx 檔案。

既然您具有 ONNX 檔案，您就可以使用以下範例，嘗試搭配此檔案執行推論：

- [使用 CNTK 搭配 ONNX 模型進行推論](#)

ONNX 教學課程

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

Chainer 到 ONNX 到 MXNet 教學課程

ONNX 概觀

[開放神經網路交換](#) (ONNX) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

本教學課程說明如何搭配 ONNX 搭配 Conda 使用深度學習 AMI。遵循下列步驟，您可以訓練模型或從一個架構載入預先訓練的模型、將此模型匯出到 ONNX，然後將此模型匯入到另一個架構。

ONNX 先決條件

若要使用此 ONNX 教學課程，您必須能夠存取具有 Conda 版本 12 或更新版本的深度學習 AMI。如需如何透過 Conda 開始使用深度學習 AMI 的詳細資訊，請參閱[使用康達深度學習 AMI](#)。

⚠ Important

這些範例使用的函數可能需要最多 8 GB 的記憶體 (或更多)。請務必選擇具有足夠記憶體的執行個體類型。

使用 Conda 使用深度學習 AMI 啟動終端工作階段，開始下列教學課程。

將 Chainer 轉換到 ONNX，然後將模型載入 MXNet

首先，啟用 Chainer 環境：

```
$ source activate chainer_p36
```

使用文字編輯器建立新的檔案，並使用指令碼中的以下程式從 Chainer model zoo 擷取模型，然後將其匯出到 ONNX 格式。

```
import numpy as np
import chainer
import chainercv.links as L
import onnx_chainer

# Fetch a vgg16 model
model = L.VGG16(pretrained_model='imagenet')

# Prepare an input tensor
x = np.random.rand(1, 3, 224, 224).astype(np.float32) * 255

# Run the model on the data
with chainer.using_config('train', False):
    chainer_out = model(x).array

# Export the model to a .onnx file
out = onnx_chainer.export(model, x, filename='vgg16.onnx')

# Check that the newly created model is valid and meets ONNX specification.
import onnx
model_proto = onnx.load("vgg16.onnx")
onnx.checker.check_model(model_proto)
```

執行此指令碼後，您將在相同的目錄中看到新建立的 .onnx 檔案。

既然您具有 ONNX 檔案，您就可以使用以下範例，嘗試搭配此檔案執行推論：

- [使用阿帕奇 MXNet \(孵化 \) 進行 ONNX 模型的推論](#)

ONNX 教學課程

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

PyTorch 前往 ONNX 轉換成 CNTK 教學

Note

從 AWS Deep Learning AMI v28 版本開始，我們不再包括 CNTK、Caffe、Caffe2 和 Theano Conda 環境。包含這些環境的先前版本將繼續提供。AWS Deep Learning AMI 不過，我們只會在有由開放原始碼社群針對這些架構發佈的安全性修正時提供這些環境的更新。

ONNX 概觀

[開放神經網路交換 \(ONNX\)](#) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

本教學課程說明如何搭配 ONNX 搭配 Conda 使用深度學習 AMI。遵循下列步驟，您可以訓練模型或從一個架構載入預先訓練的模型、將此模型匯出到 ONNX，然後將此模型匯入到另一個架構。

ONNX 先決條件

若要使用此 ONNX 教學課程，您必須能夠存取具有 Conda 版本 12 或更新版本的深度學習 AMI。如需如何透過 Conda 開始使用深度學習 AMI 的詳細資訊，請參閱[使用康達深度學習 AMI](#)。

⚠ Important

這些範例使用的函數可能需要最多 8 GB 的記憶體 (或更多)。請務必選擇具有足夠記憶體的執行個體類型。

使用 Conda 使用深度學習 AMI 啟動終端工作階段，開始下列教學課程。

將 PyTorch 模型轉換為 ONNX，然後將模型載入到 CNTK

首先，激活 PyTorch 環境：

```
$ source activate pytorch_p36
```

使用文本編輯器創建一個新文件，並在腳本中使用以下程序來訓練模擬模型 PyTorch，然後將其導出為 ONNX 格式。

```
# Build a Mock Model in Pytorch with a convolution and a reduceMean layer\  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
from torchvision import datasets, transforms  
from torch.autograd import Variable  
import torch.onnx as torch_onnx  
  
class Model(nn.Module):  
    def __init__(self):  
        super(Model, self).__init__()  
        self.conv = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3),  
stride=1, padding=0, bias=False)  
  
    def forward(self, inputs):  
        x = self.conv(inputs)  
        #x = x.view(x.size()[0], x.size()[1], -1)  
        return torch.mean(x, dim=2)  
  
# Use this an input trace to serialize the model  
input_shape = (3, 100, 100)  
model_onnx_path = "torch_model.onnx"  
model = Model()
```

```
model.train(False)

# Export the model to an ONNX file
dummy_input = Variable(torch.randn(1, *input_shape))
output = torch_onnx.export(model,
                           dummy_input,
                           model_onnx_path,
                           verbose=False)
```

執行此指令碼後，您將在相同的目錄中看到新建立的 .onnx 檔案。現在，切換到 CNTK Conda 環境來使用 CNTK 載入模型。

接著，啟用 CNTK 環境：

```
$ source deactivate
$ source activate cntk_p36
```

使用文字編輯器建立新的檔案，並使用指令碼中的以下程式在 CNTK 中開啟 ONNX 格式檔案。

```
import cntk as C
# Import the PyTorch model into CNTK via the CNTK import API
z = C.Function.load("torch_model.onnx", device=C.device.cpu(),
                   format=C.ModelFormat.ONNX)
```

執行此指令碼後，CNTK 將會載入模型。

您也可以附加以下內容到先前的指令碼然後執行，來使用 CNTK 匯出到 ONNX。

```
# Export the model to ONNX via the CNTK export API
z.save("cntk_model.onnx", format=C.ModelFormat.ONNX)
```

ONNX 教學課程

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

PyTorch 到 ONNX 轉換為 MXNet 教學課程

ONNX 概觀

[開放神經網路交換 \(ONNX\)](#) 是一種開放格式，可用來代表深度學習模型。ONNX 受到 Amazon Web Services、Microsoft、臉書和其他幾個合作夥伴的支援。您可以使用您選擇的任何架構來設計、訓練和部署深度學習模型。ONNX 模型的好處是可在架構之間輕鬆移動。

本教學課程說明如何搭配 ONNX 搭配 Conda 使用深度學習 AMI。遵循下列步驟，您可以訓練模型或從一個架構載入預先訓練的模型、將此模型匯出到 ONNX，然後將此模型匯入到另一個架構。

ONNX 先決條件

若要使用此 ONNX 教學課程，您必須能夠存取具有 Conda 版本 12 或更新版本的深度學習 AMI。如需如何透過 Conda 開始使用深度學習 AMI 的詳細資訊，請參閱[使用康達深度學習 AMI](#)。

Important

這些範例使用的函數可能需要最多 8 GB 的記憶體 (或更多)。請務必選擇具有足夠記憶體的執行個體類型。

使用 Conda 使用深度學習 AMI 啟動終端工作階段，開始下列教學課程。

將 PyTorch 模型轉換為 ONNX，然後將模型載入到 MXNet

首先，激活 PyTorch 環境：

```
$ source activate pytorch_p36
```

使用文本編輯器創建一個新文件，並在腳本中使用以下程序來訓練模擬模型 PyTorch，然後將其導出為 ONNX 格式。

```
# Build a Mock Model in PyTorch with a convolution and a reduceMean layer
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
import torch.onnx as torch_onnx
```

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3),
stride=1, padding=0, bias=False)

    def forward(self, inputs):
        x = self.conv(inputs)
        #x = x.view(x.size()[0], x.size()[1], -1)
        return torch.mean(x, dim=2)

# Use this an input trace to serialize the model
input_shape = (3, 100, 100)
model_onnx_path = "torch_model.onnx"
model = Model()
model.train(False)

# Export the model to an ONNX file
dummy_input = Variable(torch.randn(1, *input_shape))
output = torch_onnx.export(model,
                           dummy_input,
                           model_onnx_path,
                           verbose=False)
print("Export of torch_model.onnx complete!")
```

執行此指令碼後，您將在相同的目錄中看到新建立的 .onnx 檔案。現在，切換到 MXNet Conda 環境來使用 MXNet 載入模型。

接著，啟用 MXNet 環境：

```
$ source deactivate
$ source activate mxnet_p36
```

使用文字編輯器建立新的檔案，並使用指令碼中的以下程式在 MXNet 中開啟 ONNX 格式檔案。

```
import mxnet as mx
from mxnet.contrib import onnx as onnx_mxnet
import numpy as np

# Import the ONNX model into MXNet's symbolic interface
sym, arg, aux = onnx_mxnet.import_model("torch_model.onnx")
print("Loaded torch_model.onnx!")
```



```
print(sym.get_internals())
```

執行此指令碼後，MXNet 會載入模型，而且會列印一些基本模型資訊。

ONNX 教學課程

- [Apache MXNet 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 CNTK 教學課程](#)
- [Chainer 到 ONNX 到 MXNet 教學課程](#)
- [PyTorch 到 ONNX 轉換為 MXNet 教學課程](#)
- [PyTorch 前往 ONNX 轉換成 CNTK 教學](#)

模型服務

以下是使用 Conda 在深度學習 AMI 上安裝的模型服務選項。按一下其中一個選項，了解如何使用它。

主題

- [Apache MXNet 專用模型伺服器 \(MMS\)](#)
- [TensorFlow 服務](#)
- [TorchServe](#)

Apache MXNet 專用模型伺服器 (MMS)

[Apache MXNet 專用模型伺服器 \(MMS\)](#) 是一種靈活工具，用於為已從 [Apache MXNet \(incubating\)](#) 匯出，或匯出到開放神經網路交換 (ONNX) 模型格式之深度學習模型提供服務。使用 Conda 的 DLAMI 隨附預先安裝的 MMS。本教學課程適用於 MMS，示範如何為影像分類模型提供服務。

主題

- [在 MMS 上提供影像分類模型](#)
- [其他範例](#)
- [詳細資訊](#)

在 MMS 上提供影像分類模型

本教學課程說明如何使用 MMS 提供影像分類模型。模型透過 [MMS Model Zoo](#) 提供，並在您啟動 MMS 時自動下載。伺服器開始執行後，就會接聽預測請求。當您上傳影像，在此例中為一隻小貓的影

像，伺服器會從據以訓練模型的 1,000 個類別中，傳回前 5 個相符類別的預測。[MMS Model Zoo](#) 提供關於模型、其訓練方法及測試方法的詳細資訊。

在 MMS 上提供範例影像分類模型

1. 使用 Conda Connect 到深度學習 AMI 的 Amazon 彈性運算雲端 (亞馬遜 EC2) 執行個體。
2. 啟用 MXNet 環境：
 - 對於 Python 3 (含 CUDA 9.0 和 MKL-DNN) 上的 MXNet 與 Keras 2，執行此命令：

```
$ source activate mxnet_p36
```

- 對於 Python 2 (含 CUDA 9.0 和 MKL-DNN) 上的 MXNet 與 Keras 2，執行此命令：

```
$ source activate mxnet_p27
```

3. 使用下列命令執行 MMS。新增 `> /dev/null` 將在您執行進一步測試時無聲自動顯示日誌輸出。

```
$ mxnet-model-server --start > /dev/null
```

MMS 目前在您的主機上執行，並接聽推論請求。

4. 接著，使用 `curl` 命令來管理 MMS 的管理端點，並告知您想要它為哪個模型提供服務。

```
$ curl -X POST "http://localhost:8081/models?url=https%3A%2F%2Fs3.amazonaws.com%2Fmodel-server%2Fmodels%2Fsqueezenet_v1.1%2Fsqueezenet_v1.1.model"
```

5. MMS 需要知道您要使用的工作者數目。對於此測試，您可以試用 3。

```
$ curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1?min_worker=3"
```

6. 下載小貓的影像，並將它傳送到 MMS 預測端點：

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg  
$ curl -X POST http://127.0.0.1:8080/predictions/squeezenet_v1.1 -T kitten.jpg
```

預測端點會傳回 JSON 格式的預測，類似於下列前 5 個預測結果，其中的影像有 94% 的可能性包含埃及貓，隨後是 5.5% 的機會包含猓狸或美洲獅：

```
{
  "prediction": [
    [
      {
        "class": "n02124075 Egyptian cat",
        "probability": 0.940
      },
      {
        "class": "n02127052 lynx, catamount",
        "probability": 0.055
      },
      {
        "class": "n02123045 tabby, tabby cat",
        "probability": 0.002
      },
      {
        "class": "n02123159 tiger cat",
        "probability": 0.0003
      },
      {
        "class": "n02123394 Persian cat",
        "probability": 0.0002
      }
    ]
  ]
}
```

7. 多測試一些影像，或者，如果您已經完成測試，請停止伺服器：

```
$ mxnet-model-server --stop
```

本教學課程著重於基本模型服務。MMS 也支援搭配模型服務使用彈性推論。如需詳細資訊，請參閱[搭配 Amazon 彈性推論的模型服務](#)

當您準備好進一步了解其他 MMS 功能時，請參閱[上的 MMS 文件 GitHub](#)。

其他範例

彩信有各種各樣的例子，你可以在你的 DLAMI 上運行。您可以在 [MMS 專案儲存庫](#) 上檢視它們。

詳細資訊

如需更多 MMS 文件，包括如何使用 Docker 設定 MMS，或是要利用最新的 MMS 功能，請在上標示 [M MS 專案頁面](#) 的星號。GitHub

TensorFlow 服務

[TensorFlow Service](#) 是適用於機器學習模型的彈性、高效能服務系統。

tensorflow-serving-api 這是預先安裝了深度學習 AMI 與康達！您可以在 `~/examples/tensorflow-serving/` 找到訓練、匯出和提供 MNIST 模型服務的範例指令碼。

若要執行這些範例，請先使用 Conda 連線至您的深度學習 AMI，然後啟動 TensorFlow 環境。

```
$ source activate tensorflow_p37
```

現在，將目錄變更到服務範例指令碼資料夾。

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

提供預先訓練的 Inception 模型

以下是您可以嘗試以提供像是 Inception 之不同模型的範例。作為一般規則，您需要一個可服務的模型和客戶端腳本已經下載到 DLAMI。

使用 Inception 模型提供及測試推論

1. 下載模型。

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

2. Untar 模型。

```
$ unzip INCEPTION.zip
```

3. 下載哈士奇的圖片。

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

4. 啟動伺服器。請注意，若為 Amazon Linux，您必須將用於 `model_base_path` 的目錄從 `/home/ubuntu` 變更為 `/home/ec2-user`。

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

5. 當伺服器在前景執行時，您需要啟動另一個終端機工作階段才能繼續。打開一個新 TensorFlow 的終端並激活 `source activate tensorflow_p37`。然後，使用您慣用的文字編輯器來建立具有下列內容的指令碼。將其命名為 `inception_client.py`。此指令碼會以影像檔案名稱做為參數，並從預先訓練的模型取得預測結果。

```
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='Tensorflow 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. 現在執行此指令碼，將伺服器位置和連接埠以及哈士奇相片的檔案名稱當做參數傳遞。

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

訓練和提供 MNIST 模型

在本教學課程中，我們將匯出模型，然後在 `tensorflow_model_server` 應用程式中提供其服務。最後，您可以使用範例用戶端指令碼來測試模型伺服器。

執行指令碼，以便訓練和匯出 MNIST 模型。做為指令碼的唯一引數，您需要提供資料夾位置供其用來儲存模型。現在我們可以將它放在 `mnist_model` 中。指令碼會為您建立資料夾。

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

請耐心等待，因為指令碼可能需要一些時間才能提供輸出。當訓練完成且模型最後匯出時，您應該會看到下列項目：

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

下一個步驟是執行 `tensorflow_model_server` 以便提供匯出的模型做為服務。

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

提供用戶端指令碼給您測試伺服器。

若要測試，您需要開啟新的終端機視窗。

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

更多功能和範例

如果您有興趣了解有關 TensorFlow 服務的更多信息，請查看[TensorFlow 網站](#)。

您也可以將 TensorFlow 服務與 [Amazon Elastic Inference](#) 一起使用。查看有關如何[使用 Elastic Inference 與 TensorFlow 服務](#)的指南以獲取更多信息。

TorchServe

TorchServe 是一種靈活的工具，可用於提供已從匯出的深度學習模型 PyTorch。TorchServe 從 v34 開始，預先安裝了 Conda 的深度學習 AMI。

如需使用的詳細資訊 TorchServe，請參閱[模型伺服器以取得 PyTorch 文件](#)。

主題

提供影像分類模型 TorchServe

本自學課程展示如何使用提供影像分類模型 TorchServe。它使用由 PyTorch 提供的 DenseNet -161 模型。伺服器一旦執行，就會偵聽預測要求。當您上傳圖像（在這種情況下是小貓的圖像）時，伺服器返回對模型進行培訓的類中的前 5 個匹配類的預測。

若要提供範例影像分類模型 TorchServe

1. 使用 Conda v34 或更新版本的深度學習 AMI Connect 到亞馬遜彈性運算雲端 (Amazon EC2) 執行個體。
2. 啟用 pytorch_latest_p36 環境。

```
source activate pytorch_latest_p36
```

3. 克隆 TorchServe 存儲庫，然後創建一個目錄來存儲您的模型。

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

4. 使用模型歸檔器封存模型。extra-files 參數使用 TorchServe 回購中的文件，因此如有必要，請更新路徑。如需有關模型歸檔程式的詳細資訊，請參閱的[火炬模型歸檔程式](#)。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. 執行 TorchServe 以啟動端點。添加 `> /dev/null` 安靜日誌輸出。

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

6. 下載小貓的圖像並將其發送到 TorchServe 預測端點：

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

預測端點返回 JSON 的預測類似於以下前五大預測，其中圖像的可能性為 47%，其中包含埃及貓的可能性為 46%，其次是 46% 的機會，它有一隻虎斑貓。

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

7. 完成測試後，停止服務器：

```
torchserve --stop
```

其他例子

TorchServe 有各種可以在 DLAMI 實例上運行的示例。您可以在 [TorchServe 專案儲存庫範例頁面](#) 上檢視它們。

更多信息

如需更多 TorchServe 文件，包括如何設定 TorchServe 的 Docker 和最新 TorchServe 功能，請參閱 [GitHub 的 TorchServe 專案頁面](#)。

升級您的 DLAMI

在這裡，您可以找到有關升級 DLAMI 的信息以及更新 DLAMI 軟件的提示。

主題

- [升級到新的 DLAMI 版本](#)
- [軟體更新的秘訣](#)

升級到新的 DLAMI 版本

DLAMI 的系統映像會定期更新，以利用新的深度學習架構版本、CUDA 和其他軟體更新，以及效能調整。如果您已經使用 DLAMI 一段時間，並且想要利用更新，則需要啟動新的執行個體。您也必須手動傳輸任何資料集、檢查點或其他寶貴的資料。相反地，您可以使用 Amazon EBS 保留您的資料，並將其附加到新的 DLAMI。利用這種方式，您可以經常升級，同時減少轉換資料所需的時間。

Note

在 DLAMI 之間連接和移動 Amazon EBS 磁碟區時，您必須同時在相同的可用區域中同時擁有 DLAMI 和新磁碟區。

1. 使用亞馬遜 EC2 主控台建立新的亞馬遜 EBS 磁碟區。如需詳細指示，請參閱[建立 Amazon EBS 磁碟區](#)。
2. 將新建立的 Amazon EBS 磁碟區連接至 DLAMI。如需詳細指示，請參閱[附加 Amazon EBS 磁碟區](#)。
3. 傳輸您的資料，例如資料集、檢查點和組態檔案。
4. 啟動一個 DLAMI。如需詳細指示，請參閱[啟動和設定 DLAMI](#)。
5. 從舊的 DLAMI 中分離亞馬遜 EBS 卷。如需詳細指示，請參閱[分離 Amazon EBS 磁碟區](#)。
6. 將 Amazon EBS 磁碟區連接至 DLAMI 依照步驟 2 的指示連接磁碟區。
7. 驗證新 DLAMI 上的數據可用後，請停止並終止舊的 DLAMI。如需詳細的清除說明，請參閱[清除](#)。

軟體更新的秘訣

有時，您可能需要手動更新 DLAMI 上的軟件。通常建議您使用 pip 來更新 Python 套件。您也應該使 pip 用在 Conda 的深度學習 AMI 上更新 Conda 環境中的套件。請參閱特定架構或軟體網站的升級和安裝說明。

Note

我們無法保證套件更新會成功。嘗試在具有不相容相依性的環境中更新套件可能會導致失敗。在這種情況下，您應該聯繫庫維護者以查看是否可以更新軟件包依賴關係。或者，您也可以嘗試以允許更新的方式修改環境。不過，這項修改可能意味著移除或更新現有套件，這表示我們無法再保證此環境的穩定性。

如果您有興趣運行特定軟件包的最新主分支，請激活適當的環境，然後添加 `--pre` 到 `pip install --upgrade` 命令的末尾。例如：

```
source activate mxnet_p36
pip install --upgrade mxnet --pre
```

AWS Deep Learning AMI 隨附許多 Conda 環境以及許多預先安裝的套件。由於預先安裝的套件數量，很難找到一組保證相容的套件。您可能會看到警告「環境不一致，請仔細檢查包裝計劃」。DLAMI 可確保所有 DLAMI 提供的環境都正確無誤，但無法保證任何使用者安裝的套件都能正常運作。

中的安全性 AWS Deep Learning AMI

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，該架構專為滿足對安全性最敏感的組織的需求而打造。

安全是 AWS 與您之間共同承擔的責任。[共同責任模型](#)將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 — AWS 負責保護在 AWS 雲端中執行 AWS 服務的基礎架構。AWS 還為您提供可以安全使用的服務。若要瞭解適用於 DLAMI 的法規遵循計劃，請參閱[合規計劃AWS 服務範圍內的 AWS 服務範圍 \(遵\)](#)。
- 雲端中的安全性 — 您的責任取決於您使用的 AWS 服務。您也必須對其他因素負責，包括資料的機密性、您的公司的要求和適用法律和法規。

本文件可協助您瞭解如何在使用 DLAMI 時套用共同責任模型。下列主題說明如何設定 DLAMI 以符合安全性和合規性目標。您還將學習如何使用其他 AWS 服務來幫助您監控和保護 DLAMI 資源。

如需詳細資訊，請參閱 [Amazon EC2 中的安全性](#)。

主題

- [資料保護 AWS Deep Learning AMI](#)
- [中的 Identity and Access Management AWS Deep Learning AMI](#)
- [登錄和監控 AWS Deep Learning AMI](#)
- [符合性驗證 AWS Deep Learning AMI](#)
- [韌性 AWS Deep Learning AMI](#)
- [基礎架構安全 AWS Deep Learning AMI](#)

資料保護 AWS Deep Learning AMI

AWS [共同責任模型](#)適用於 AWS 深度學習 AMI 中的資料保護。如此模型中所述，AWS 負責保護執行所有 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，我們建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 登入資料並設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源進行通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用設定 API 和使用者活動記錄 AWS CloudTrail。
- 使用 AWS 加密解決方案以及其中的所有默認安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取時需要經 AWS 過 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-2 概觀](#)。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用控制台，API 或 SDK AWS 服務使用 DLAMI 或 AWS 其他方式時。AWS CLI您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

中的 Identity and Access Management AWS Deep Learning AMI

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。AWS 服務 IAM 管理員控制哪些人可以驗證 (登入) 和授權 (具有權限) 以使用 DLAMI 資源。IAM 是您可以使用的 AWS 服務，無需額外付費。

如需 Identity and Access Management 的詳細資訊，請參閱 [Amazon EC2 中的 Identity and Access Management](#)。

主題

- [使用身分來驗證](#)
- [使用政策管理存取權](#)
- [IAM 搭配 Amazon EMR](#)

使用身分來驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》AWS 帳戶中的[如何登入](#)您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的[簽署 AWS API 請求](#)。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。如需更多資訊，請參閱 AWS IAM Identity Center 使用者指南中的[多重要素驗證](#)和 IAM 使用者指南中的[在 AWS 中使用多重要素驗證 \(MFA\)](#)。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的[需要根使用者憑證的任務](#)。

IAM 使用者和群組

[IAM 使用者](#)是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 [IAM 使用者指南](#) 中的為需要長期憑證的使用案例定期輪換存取金鑰。

[IAM 群組](#)是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱 IAM 使用者指南中的[建立 IAM 使用者 \(而非角色\) 的時機](#)。

IAM 角色

[IAM 角色](#)是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以[切換角色，在中暫時擔任 IAM 角色](#)。AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法更多相關資訊，請參閱 IAM 使用者指南中的[使用 IAM 角色](#)。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 若要向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱 [IAM 使用者指南](#) 中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的 [許可集](#)。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源 (而不是使用角色作為代理)。若要了解跨帳戶存取權角色和資源型政策間的差異，請參閱 IAM 使用者指南中的 [IAM 角色與資源類型政策的差異](#)。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
- 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱 [《轉發存取工作階段》](#)。
- 服務角色 – 服務角色是服務擔任的 [IAM 角色](#)，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需更多資訊，請參閱 IAM 使用者指南中的 [建立角色以委派許可給 AWS 服務](#)。
- 服務連結角色 — 服務連結角色是連結至 AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中，AWS 帳戶 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色來管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需更多資訊，請參閱 IAM 使用者指南中的 [利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式](#)。

若要了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的 [建立 IAM 角色 \(而非使用者\) 的時機](#)。

使用政策管理存取權

您可以透過 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS，當與身分識別或資源相關聯時，會定義其權限。AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的更多相關資訊，請參閱 IAM 使用者指南中的 [JSON 政策概觀](#)。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的 [建立 IAM 政策](#)。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶。受管政策包括 AWS 受管政策和客戶管理的策略。若要了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的 [在受管政策和內嵌政策間選擇](#)。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中 [指定主體](#)。主參與者可以包括帳戶、使用者、角色、同盟使用者或。AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些委託人 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 和 Amazon VPC 是支援 ACL 的服務範例。AWS WAF 若要進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的 [存取控制清單 \(ACL\) 概觀](#)。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 Principal 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可範圍的更多相關資訊，請參閱 IAM 使用者指南中的 [IAM 實體許可範圍](#)。
- 服務控制策略 (SCP) — SCP 是 JSON 策略，用於指定中組織或組織單位 (OU) 的最大權限。AWS Organizations 是一種用於分組和集中管理您企業擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制策略 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需組織和 SCP 的更多相關資訊，請參閱 AWS Organizations 使用者指南中的 [SCP 運作方式](#)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需更多資訊，請參閱 IAM 使用者指南中的 [工作階段政策](#)。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的 [政策評估邏輯](#)。

IAM 搭配 Amazon EMR

您可以 AWS Identity and Access Management 搭配 Amazon EMR 來定義使用者、AWS 資源、群組、角色和政策。您也可以控制這些使用者和角色可存取的 AWS 服務。

如需將 IAM 與 Amazon EMR 搭配使用的詳細資訊，請參閱 Amazon EMR 的 [AWS Identity and Access Management](#)。

登錄和監控 AWS Deep Learning AMI

您的 AWS Deep Learning AMI 執行個體隨附數個 GPU 監控工具，其中包括向 Amazon 報告 GPU 使用率統計資料的公用程式 CloudWatch。如需詳細資訊，請參閱 [GPU 監控和最佳化](#) 及 [監控 Amazon EC2](#)。

使用量追蹤

下列 AWS Deep Learning AMI 作業系統發行版包含可 AWS 收集執行個體類型、執行個體 ID、DLAMI 類型和作業系統資訊的程式碼。不會收集或保留 DLAMI 中使用的指令的相關資訊。不會收集或保留任何其他有關「DLAMI 院」的資料。

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

若要選擇退出 DLAMI 的使用狀況追蹤，請在啟動期間將標籤新增至 Amazon EC2 執行個體。標籤應該使用關聯值設定為的鍵 `OPT_OUT_TRACKING` 為 `true`。如需詳細資訊，請參閱 [標記您的 Amazon EC2 資源](#)。

符合性驗證 AWS Deep Learning AMI

協力廠商稽核人員會評估其安全性與合規性，AWS Deep Learning AMI 做為多個 AWS 合規計畫的一部分。如需支援的合規計畫相關資訊，請參閱 [Amazon EC2 的合規驗證](#)。

如需特定規範計劃範圍內的 AWS 服務清單，請參閱 [合規計劃 AWS 服務範圍](#) 方案)。如需一般資訊，請參閱 [AWS 規範計劃 AWS](#)。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱 [下載 AWS Artifact](#) 的報表。

您在使用 DLAMI 時的合規責任取決於您資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- [安全與合規快速入門指南](#)：這些部署指南討論架構考量，並提供在 AWS 上部署以安全及合規為重心之基準環境的步驟。
- [AWS 合規資源 AWS](#) — 此工作簿和指南集合可能適用於您的產業和所在地。

- [使用AWS Config 開發人員指南中的規則評估資源](#) — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#)— 此 AWS 服務提供安全狀態的全面檢視，協助您檢查您 AWS 是否符合安全性產業標準和最佳做法。

韌性 AWS Deep Learning AMI

AWS 全球基礎架構是圍繞區 AWS 域和可用區域建立的。AWS 區域提供多個實體分離和隔離的可用區域，這些區域透過低延遲、高輸送量和高度備援的網路連線。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需區域和可用區域的相關 AWS 資訊，請參閱[AWS 全域基礎結構](#)。

如需協助支援資料備援和備份需求相關功能的詳細資訊，請參閱[Amazon EC2 中的備援](#)。

基礎架構安全 AWS Deep Learning AMI

的基礎設施安全性 AWS Deep Learning AMI 由 Amazon EC2 提供支援。如需詳細資訊，請參閱[Amazon EC2 中的基礎設施安全](#)。

DLAMI 的重要變化

常見問答集

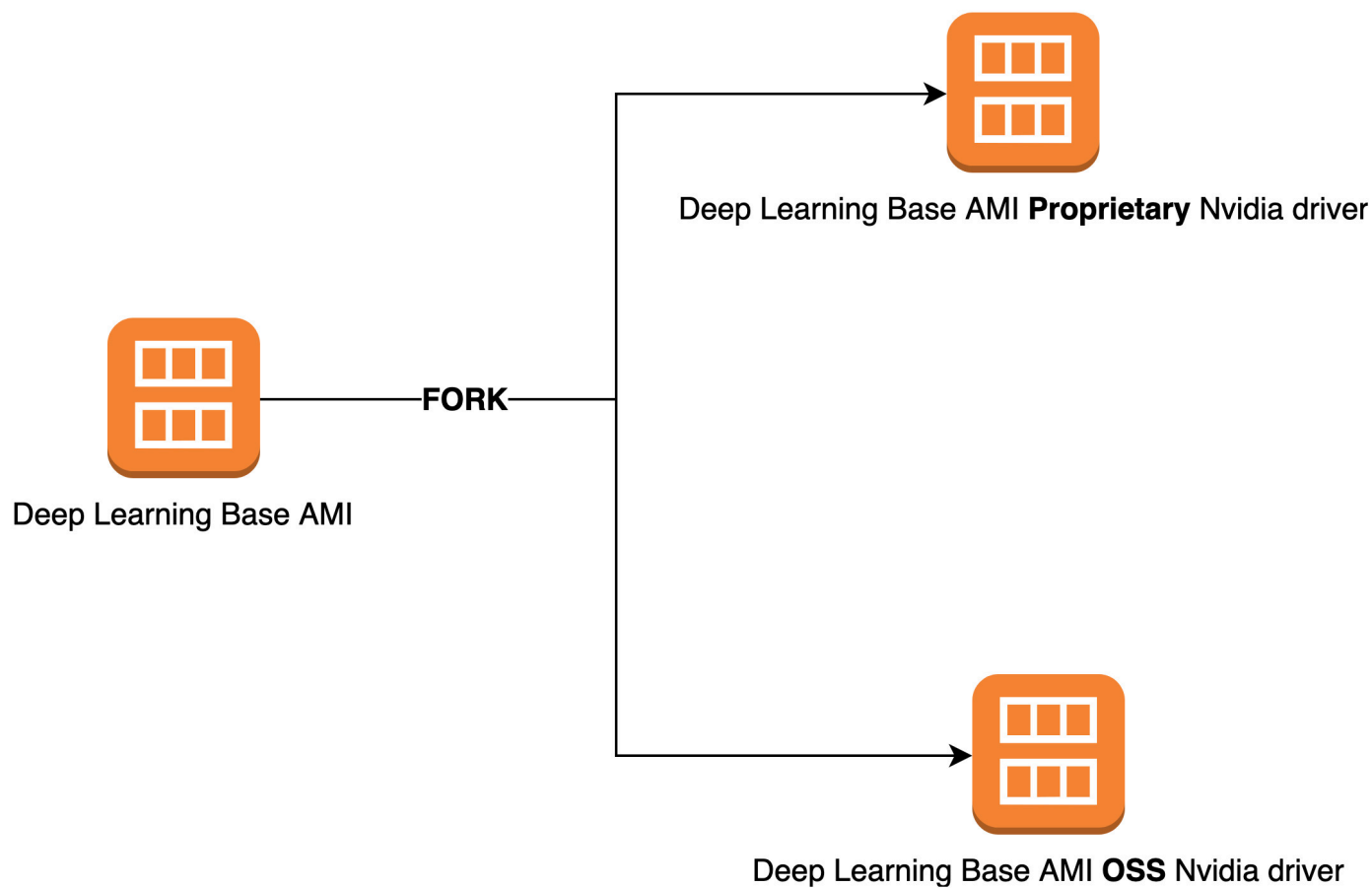
- [有什麼變化？](#)
- [為什麼需要進行此更改？](#)
- [哪些 DLAM 會受到此變更的影響？](#)
- [這對你意味著什麼？](#)
- [你應該什麼時候開始使用新的 DLAMI？](#)
- [新的 DLAMI 會否有任何功能損失？](#)
- [關於可攜式資料中心呢？](#)

有什麼變化？

在二零二三年十一月十五日 AWS Deep Learning AMI (中文版) 將會分為兩個不同的組別：

- 使用 Nvidia 專有驅動程序的 DLAM (支持 P3 , P3DN , G3)。
- 使用 Nvidia 操作系統驅動程序的 DLAM (支持 G4DN , G5 , P4 , P5)。

因此，將為兩個類別中的每個類別建立新的 DLAMI，其中包含新名稱和新 AMI ID。這些 DLAMI 無法互換-即來自一個群組的 DLAMI 將不支援另一個群組支援的執行個體，例如支援 p5 的 DLAMI 將不支援 g3，反之亦然。



為什麼需要進行此更改？

目前用於 NVIDIA GPU 的 DLAMI 包括來自 NVIDIA 的專有內核驅動程序。不過，最近上游的 Linux 核心社群接受了一項變更，該變更會將專有核心驅動程式 (例如 NVIDIA GPU 驅動程式) 與其他核心驅動程式通訊隔離開來。這項變更會停用 P4/P5 系列執行個體上的 GPUDirect RDMA，這項機制可讓 GPU 有效地使用 EFA 進行分散式訓練。因此，DLAMI 將使用 OpenRM 驅動程序 (NVIDIA 開源驅動程序)，與開源的 EFA 驅動程序鏈接到支持 G4DN，G5，P4 和 P5。不過，這個 OpenRM 驅動程式不支援較舊的執行個體 (P3、G3 等) 因此，為了確保我們持續提供目前、高效能和安全的 DLAMI 支援這兩種類型的執行個體，我們會將 DLAMI 分成兩個群組-一個使用 OpenRM 驅動程式 (支援 G4dn、G5、P4 和 P5)，另一個使用較舊的專有驅動程式 (支援較舊的執行個體 P3、P3DN、G3)。

哪些 DLAM 會受到此變更的影響？




所有 DLAMI 都會受到此變更的影響。

這對你意味著什麼？

新的 DLAMI 將繼續提供目前 DLAMI 的功能、效能和安全性，只要它們是在相容的執行個體類型上執行。如果您使用 DLAMI，則需要確保在每個 DLAMI 的發行說明中提到的其中一個兼容實例上啟動 DLAMI (請參閱此處)。例如：您需要適應此變更，才能：

- 使用正確的 CLI 查詢呼叫 DLAMI (請參閱下文)
- 從主控台和 CLI 在相容的執行個體類型上啟動 DLAMI

如果您是從 EC2 主控台快速入門啟動 DLAMI：每個 DLAMI 說明都會列出 EC2 主控台支援的執行個體類型。您應該在相容的執行個體上啟動 DLAMI。

 ubuntu Verified provider	Deep Learning Base GPU AMI (Ubuntu 20.04) 20231018 ami-05f9aedeafddcf112 (64-bit (x86)) Supported EC2 instances: P5*, P4*, P3*, G3*, G5*, G4dn. Release notes: https://aws.amazon.com/releases/notes/aws-deep-learning-base-gpu-ami-ubuntu-20-04/ Platform: ubuntu Root device type: ebs	Select <input checked="" type="radio"/> 64-bit (x86)
 ubuntu Verified provider	Deep Learning AMI GPU PyTorch 2.0.1 (Ubuntu 20.04) 20231003 ami-005656037407fcf99 (64-bit (x86)) Supported EC2 instances: P5, P4d, P4de, P3, P3dn, G5, G4dn, G3. Release notes: https://docs.aws.amazon.com/dlami/latest/devguide/appendix-ami-release-notes.html Platform: ubuntu Root device type: ebs	Select <input checked="" type="radio"/> 64-bit (x86)
 ubuntu Verified provider	Deep Learning AMI Neuron PyTorch 1.13 (Ubuntu 20.04) 20231003 ami-0f337e1c69255b2b6 (64-bit (x86)) Supported EC2 instances: inf1, Trn1, Trn1n, inf2. Release notes: https://docs.aws.amazon.com/dlami/latest/devguide/appendix-ami-release-notes.html Platform: ubuntu Root device type: ebs	Select <input checked="" type="radio"/> 64-bit (x86)

如果您使用 CLI 啟動 DLAMI，則必須修改您的查詢。例如：

目前，下列 CLI 查詢用於支援所有執行個體 [P3、P3dn、G3、G4 DN、G5、P4、P5] 的基礎 DLAMI：

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

新的 CLI 查詢將是：

對於支援 P3、P3dn 和 G3 的 DLAMI 本數位數據：

```
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
```

對於支援 G4 DN、G5、P4 和 P5 的基本數據：

```
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
```

請在[這裡](#)參閱新 AMI 的更新版本說明。如需如何在 EC2 執行個體上啟動 AMI，請參閱[這裡](#)的指示。

你應該什麼時候開始使用新的 DLAMI？

您應該儘快針對最新的架構、相依性、修補程式和功能開始使用新的 DLAMI。[或者，如果您使用的是 2023 年 11 月 8 日之前發布的 Amazon Linux 2 DLAMI，那麼您可以選擇繼續實時修補他們的 DLAMI \(請參閱此處的說明\)，直到 2023 年 11 月 30 日為止。](#)

新的 DLAMI 會否有任何功能損失？

否，新的 DLAMI 不會損失任何功能。拆分後的新 DLAMI 將繼續提供舊 DLAMI 的所有功能，性能和安全性，只要它們是在兼容的實例上運行。我們將 DLAMI 分成兩個群組，以便繼續提供最新、高效能且安全的 DLAMI，供您在廣泛的執行個體上使用。

關於可攜式資料中心呢？

DLC 不包含 NVIDIA 驅動程式，因此它們不會受到此變更的影響。但是您應該確定 DLC 是在與基礎執行個體相容的 AMI 上執行。

相關資訊

主題

- [論壇](#)
- [相關 部落格文章](#)
- [常見問答集](#)

論壇

- [論壇：AWS深度學習 AMI](#)

相關 部落格文章

- [更新了與深度學習 AMI 相關的文章列表](#)
- [啟動一個AWS Deep Learning AMI \(在 10 分鐘內 \)](#)
- [在 Amazon EC2 C5 和 P3 執行個體上使用最佳化 TensorFlow 1.6 , 加快訓練速度](#)
- [適用於 Machine Learning 從業人員的新AWS深度學習 AMI](#)
- [新推出的訓練課程：Machine Learning 與深度學習簡介AWS](#)
- [深度學習之旅AWS](#)

常見問答集

- 問：如何追蹤與 DLAMI 相關的產品公告？

以下是兩個建議：

- [將此部落格類別加入書籤，可在此找到「AWS深度學習 AMI」：更新深度學習 AMI 相關文章清單。](#)
- 「觀看」[論壇：AWS深度學習 AMI](#)
- 問：是否已安裝 NVIDIA 驅動程式和 CUDA？

是。一些 DLAMI 有不同的版本。具[使用康達深度學習 AMI](#)有任何 DLAMI 的最新版本。[CUDA 安裝和架構連結](#)中提供更多詳細資訊。您也可以參考特定 AMI 的版本說明，以確認已安裝的內容。

- 問：是否已安裝 CUDNN？

是。

- 問：如何查看已偵測到 GPU 及其目前狀態？

執行 `nvidia-smi`。這會顯示一個或多個 GPU (取決於執行個體類型) 及其目前的記憶體耗用量。

- 問：是否已為我設定虛擬環境？

是，但僅限 [使用康達深度學習 AMI](#)。

- 問：安裝的是什麼版本的 Python？

每個 DLAMI 都有蟒蛇 2 和 3。 [使用康達深度學習 AMI](#) 對於每個架構版本都已具備環境。

- 問：是否已安裝 Keras？

取決於 AMI。 [使用康達深度學習 AMI](#) 在每個架構的前端提供 Keras。 Keras 版本取決於架構的支援。

- 問：它是免費的嗎？

所有的 DLAMI 都是免費的。不過，取決於您選擇的執行個體類型，執行個體可能不是免費的。如需更多詳細資訊，請參閱 [特殊護理 DLAMI 價](#)。

- 問：我收到架構中的 CUDA 錯誤或 GPU 相關訊息。怎麼回事？

檢查您使用哪一種執行個體類型。它需有 GPU，許多範例和教學課程才能運作。如果執行中沒有 `nvidia-smi` 顯示 GPU，則您需要使用具有一或多個 GPU 的執行個體啟動另一個 DLAMI。如需更多詳細資訊，請參閱 [選取 DLAMI 的例證類型](#)。

- 問：我可以使用泊塢視窗嗎？

自從使用 Conda 的深度學習 AMI 版本 14 以來，Docker 就已預先安裝。請注意，您需要在 GPU 執行個體上使用 [nvidia-docker](#) 來使用 GPU。

- 問：Linux 區域有哪些區域可供使用？

區域	代碼
美國東部 (俄亥俄)	us-east-2
美國東部 (維吉尼亞北部)	us-east-1
GovCloud	us-gov-west-1

區域	代碼
美國西部 (加利佛尼亞北部)	us-west-1
美國西部 (奧勒岡)	us-west-2
北京 (中國)	cn-north-1
寧夏 (中國)	cn-northwest-1
亞太區域 (孟買)	ap-south-1
亞太區域 (首爾)	ap-northeast-2
亞太區域 (新加坡)	ap-southeast-1
亞太區域 (雪梨)	ap-southeast-2
亞太區域 (東京)	ap-northeast-1
加拿大 (中部)	ca-central-1
歐洲 (法蘭克福)	eu-central-1
歐洲 (愛爾蘭)	eu-west-1
歐洲 (倫敦)	eu-west-2
歐洲 (巴黎)	eu-west-3
南洲 (聖保羅)	sa-east-1

- 問：視窗 DLAM 在哪些區域可供使用？

區域	代碼
美國東部 (俄亥俄)	us-east-2
美國東部 (維吉尼亞北部)	us-east-1

區域	代碼
GovCloud	us-gov-west-1
美國西部 (加利佛尼亞北部)	us-west-1
美國西部 (奧勒岡)	us-west-2
北京 (中國)	cn-north-1
亞太區域 (孟買)	ap-south-1
亞太區域 (首爾)	ap-northeast-2
亞太區域 (新加坡)	ap-southeast-1
亞太區域 (雪梨)	ap-southeast-2
亞太區域 (東京)	ap-northeast-1
加拿大 (中部)	ca-central-1
歐洲 (法蘭克福)	eu-central-1
歐洲 (愛爾蘭)	eu-west-1
歐洲 (倫敦)	eu-west-2
歐洲 (巴黎)	eu-west-3
南洲 (聖保羅)	sa-east-1

特拉米的發行公告

Note

AWS Deep Learning AMI每晚有安全性修補程式的發行節奏。這些增量安全性修補程式不包含在官方發行說明中。

請參考 [DLAMI Support 原則頁面](#)，瞭解任何不受支援的架構版本說明。

基地 DLAMI

GPU

- [AWS 深度學習基礎 AMI \(Amazon Linux 2\)](#)
- [AWS 深度學習基礎 AMI](#)

AWS 神經元

- [AWS 深度學習基礎 AMI 神經元 \(Amazon Linux 2\)](#)
- [AWS 深度學習基礎 AMI 經元](#)

高通公司

- [AWS 深度學習基礎高通 AMI \(Amazon Linux 2 \)](#)

單框架 DLAMI

PyTorch特定 AMI

- GPU
 - [AWS 深度學習網路 GPU PyTorch 2.1](#)
 - [AWS 深度學習 AMI GPU PyTorch 1.13 \(Amazon Linux 2\)](#)
 - [AWS 深度學習 AMI GPU PyTorch](#)

- AWS 神经元

- [AWS 深度學習 AMI 神經元 PyTorch 1.13 \(Amazon Linux 2\)](#)
- [AWS 深度學習 AMI 神經元 PyTorch](#)

TensorFlow 特定 AMI

- GPU


- [AWS 深度學習 AMI GPU TensorFlow 2.15 \(Amazon Linux 2\)](#)
- [AWS 深度學習 AMI GPU TensorFlow](#)
- [AWS 深度學習 AMI GPU TensorFlow 2.13 \(Amazon Linux 2\)](#)
- [AWS 深度學習 AMI GPU TensorFlow](#)

- AWS 神经元

- [AWS 深度學習 AMI 神經元 TensorFlow 2.10 \(Amazon Linux 2\)](#)
- [AWS 深度學習 AMI 神經元 TensorFlow](#)

多框架

GPU

 Note

如果您只使用一個機器學習架構，我們建議您使用 [單框架 DLAMI](#)

- [AWS 深度學習 AMI \(Amazon 2\)](#)

AWS 神经元

- [AWS 深度學習 AMI 經元](#)

DLAMI 的通知

下表列出 AWS Deep Learning AMI 中已取代的功能的相關資訊。

已取代的功能	取代日期	取代通知
Ubuntu 16.04	10/07/2021	Ubuntu Linux 16.04 LTS 於 2021 年 4 月 30 日達到了其五年 LTS 窗口的結束，並且不再受其供應商的支持。截至 2021 年 10 月，新版本中已不再對深度學習基礎 AMI (Ubuntu 16.04) 進行更新。以前的版本將繼續可用。
Amazon Linux	10/07/2021	亞馬遜 Linux 是 end-of-life 截至 2020 年十二月的資料。截至 2021 年 10 月，新版本中不再有深度學習 AMI (亞馬遜 Linux) 的更新。舊版本的 AMI (Amazon Linux) 將繼續用的舊版本。
Chainer	2020/7/1	Chainer 已宣布自 2019 年 12 月起 終止主要版本 。因此，從 2020 年 7 月開始，我們將不再在 DLAMI 上包含 Chainer Conda 環境。包含這些環境的舊版 DLAMI 將繼續提供。只有在有由開放原始碼社群針對這些架構發佈的安全性修正時，我們才會提供這些環境的更新。
Python 3.6	2020/6/15	由於客戶請求，針對新的 TF/MX/PT 版本，我們即將移轉至 Python 3.7。

已取代的功能	取代日期	取代通知
Python 2	2020/1/1	<p>Python 開放原始碼社群已正式終止支援 Python 2。</p> <p>TensorFlow PyTorch, 和 MXNet 社群也宣布，TensorFlow 1.15、TensorFlow 2.1、PyTorch 1.4 和 MXNet 1.6.0 版本將是最後一個支援 Python 2 的版本。</p>

AWS Deep Learning AMI 開發人員指南的文件歷史記錄

變更	描述	日期
重力子 DLAMI	AWS Deep Learning AMI現在支援以 Arm 處理器為基礎的重力子 GPU 上的影像。	2021 年 11 月 29 日
哈瓦那·DLAMI	AWS Deep Learning AMI現在支持哈瓦那高迪硬件和哈瓦那突觸 SDK。	2021 年 10 月 25 日
TensorFlow 2	與康達的深度學習 AMI 現在隨附 TensorFlow 2 與 CUDA 10。	2019 年 12 月 3 日
AWS推論	深度學習 AMI 現在支援AWS推論硬體和AWS神經元 SDK。	2019 年 12 月 3 日
搭配成立模型使用 TensorFlow 服務	針對「Elastic Inference」和「彈性推論」的「TensorFlow 服務」新增了一個將推論與 Inception 模型搭配使用的範例。	2018 年 11 月 28 日
使用 256 個 GPU TensorFlow 和霍洛沃德進行訓練	已更新「TensorFlow 與 Horovod」教學課程，以新增多節點訓練的範例。	2018 年 11 月 28 日
Elastic Inference	已在安裝指南中加入彈性的推論事前準備和相關資訊。	2018 年 11 月 28 日
彩信 v1.0 發布在 DLAMI。	MMS 教學課程已更新為使用新的模型存檔格式 (.mar)，以及示範新的啟動和停止功能。	2018 年 11 月 15 日
TensorFlow 從夜間構建安裝	已新增教學課程，內容涵蓋如何解除安裝 TensorFlow，然後	2018 年 10 月 16 日

	使用 Conda 在深度學習 AMI TensorFlow 上安裝夜間組建。	
從夜間組建安裝 CNTK	已新增教學課程，內容涵蓋如何解除安裝 CNTK，然後使用 Conda 在深度學習 AMI 上安裝 CNTK 的夜間組建。	2018 年 10 月 16 日
從夜間構建安裝阿帕奇 MXNet (孵化)	已新增教學課程，內容涵蓋如何解除安裝 MXNet，然後使用 Conda 在深度學習 AMI 上安裝 MXNet 的夜間組建。	2018 年 10 月 16 日
PyTorch 從夜間構建安裝	已新增教學課程，內容涵蓋如何解除安裝 PyTorch，然後使用 Conda 在深度學習 AMI PyTorch 上安裝夜間組建。	2018 年 9 月 25 日
泊塢窗現在已預先安裝在您的 DLAMI 上	自從 14 版的深度學習 AMI 與孔達，碼頭工人和 NVIDIA 的 GPU 泊塢窗版本已預先安裝。	2018 年 9 月 25 日
TensorBoard 教程	範例已移到 ~/examples/tensorboard。教學課程路徑已更新。	2018 年 7 月 23 日
MXBoard 教學課程	新增如何使用 MXBoard 視覺化 MXNet 模型的教學課程。	2018 年 7 月 23 日
分散式訓練教學	新增如何使用 Keras-MXNet 進行多重 GPU 訓練的教學課程。Chainer 的教學課程已更新至 v4.2.0。	2018 年 7 月 23 日
康達教程	範例 MOTD 已更新，以反映較新版本。	2018 年 7 月 23 日
鏈接教程	教學課程已更新為使用 Chainer 來源的最新範例。	2018 年 7 月 23 日

舊版更新：

下表說明 2018 年 7 月前 AWS Deep Learning AMI 每個版本的重要變更。

變更	描述	日期
TensorFlow 與霍洛沃德	新增了與霍洛沃德— ImageNet 起 TensorFlow 訓練的教學課程。	2018 年 6 月 6 日
升級指南	新增升級指南。	2018 年 5 月 15 日
新的區域和新的 10 分鐘教學課程	新增新的區域：美國西部 (加州北部)、南美洲、加拿大 (中部)、歐洲 (倫敦) 和歐洲 (巴黎)。此外，10 分鐘教學課程的第一個版本標題為：「深度學習 AMI 入門」。	2018 年 4 月 26 日
Chainer 教學課程	新增在多重 GPU、單一 GPU 和 CPU 模式中使用 Chainer 的教學課程。CUDA 整合已從 CUDA 8 升級到 CUDA 9，適用於數個架構。	2018 年 2 月 28 日
Linux AMI 版本 3.0 版，以及 MXNet 模型伺服器、TensorFlow 服務和 TensorBoard	針對 Conda AMI 新增教學課程，其中包含使用 MXNet 模型伺服器 v0.1.5、TensorFlow 服務 v1.4.0 和 v0.4.0 的新模型和 TensorBoard 視覺化服務功能。Conda 和 CUDA 概觀中描述 AMI 和架構 CUDA 功能。最新版本備註移到 https://aws.amazon.com/releasenotes/	2018 年 1 月 25 日
Linux AMI v2.0	基礎、來源和 Conda AMI 更新為使用 NCCL 2.1。來源和康達 AMI 更新為 MXNet 1.0	2017 年 12 月 11 日

變更	描述	日期
	版、PyTorch 0.3.0 和克拉斯 2.0.9。	
新增兩個 Windows AMI 選項	發佈Windows 2012 R2 和 2016 AMI：新增到 AMI 選購指南並新增到版本備註。	2017 年 11 月 30 日
初始文件版本	詳細描述變更，並提供已變更主題/章節的連結。	2017 年 11 月 15 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 [AWS 詞彙表](#)。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。