



使用 Amazon Q Developer 產生內嵌和助理程式碼的最佳實務

AWS 方案指引



AWS 方案指引: 使用 Amazon Q Developer 產生內嵌和助理程式碼的最佳實務

Copyright © 2026 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

簡介	1
目標	1
開發人員工作流程	3
設計和規劃	4
編碼	4
程式碼檢閱	4
整合和部署	5
進階功能	6
Amazon Q 開發人員程式碼轉換	6
Amazon Q 開發人員自訂	6
最佳編碼實務	8
加入的最佳實務	8
Amazon Q Developer 的先決條件	8
使用 Amazon Q Developer 時的最佳實務	8
Amazon Q Developer 中的資料隱私權和內容使用量	9
程式碼產生最佳實務	9
程式碼建議的最佳實務	10
程式碼範例	12
Python 範例	12
產生類別和函數	12
文件程式碼	13
產生演算法	14
產生單位測試	17
Java 範例	18
產生類別和函數	18
文件程式碼	21
產生演算法	23
產生單位測試	25
聊天範例	26
詢問 AWS 服務	26
產生程式碼	27
產生單位測試	29
說明程式碼	31
疑難排解	34

產生空的程式碼	34
持續註解	35
不正確的內嵌程式碼產生	36
聊天結果不足	41
FAQs	45
什麼是 Amazon Q Developer?	45
如何存取 Amazon Q Developer?	45
Amazon Q Developer 支援哪些程式設計語言?	45
如何向 Amazon Q Developer 提供內容，以獲得更好的程式碼產生?	45
如果使用 Amazon Q Developer 產生內嵌程式碼不準確，該怎麼辦?	45
如何使用 Amazon Q Developer 聊天功能進行程式碼產生和疑難排解?	46
使用 Amazon Q Developer 有哪些最佳實務?	46
我可以自訂 Amazon Q Developer 來根據自己的程式碼產生建議嗎?	46
後續步驟	47
資源	48
AWS 部落格	48
AWS 文件	48
AWS 研討會	48
貢獻者	49
文件歷史紀錄	50
詞彙表	51
#	51
A	51
B	54
C	55
D	58
E	61
F	63
G	64
H	65
I	66
L	68
M	69
O	73
P	75
Q	77

R	77
S	80
T	83
U	84
V	84
W	85
Z	86
.....	lxxxvii

使用 Amazon Q Developer 產生內嵌和助理程式碼的最佳實務

Amazon Web Services ([貢獻者](#))

2024 年 8 月 ([文件歷史記錄](#))

傳統上，開發人員倚賴來自各種來源的專業知識、文件和程式碼片段來撰寫和維護程式碼。雖然這些方法能為業界提供良好的服務，但它們可能耗時且容易發生人為錯誤，進而導致效率低下和潛在的錯誤。

這是 Amazon Q Developer 在中採取的步驟，以改善開發人員的旅程。Amazon Q Developer 是強大的 AWS 生成式 AI 支援助理，旨在透過提供智慧程式碼產生和建議來加速程式碼開發任務。

不過，如同任何新技術一樣，可能會有挑戰。不切實際的期望、加入困難、疑難排解不正確的程式碼產生，以及正確使用 Amazon Q 功能，都是開發人員可能面臨的常見障礙。此完整指南解決了這些挑戰，提供實際案例、詳細的最佳實務、故障診斷，以及專門針對 Python 和 Java 的實用實際程式碼範例，這是兩種最廣泛採用的程式設計語言。

本指南著重於使用 Amazon Q Developer 執程式碼開發任務，例如：

- 程式碼完成 – 即時產生內嵌建議做為開發人員程式碼。
- 程式碼改善和建議 – 討論軟體開發、產生自然語言的新程式碼，以及改善現有的程式碼。

目標

本指南的目標是支援 Amazon Q Developer 的新使用者或持續使用者的開發人員，協助他們在日常編碼任務中成功使用該服務。開發團隊經理也可以從閱讀本指南中受益。

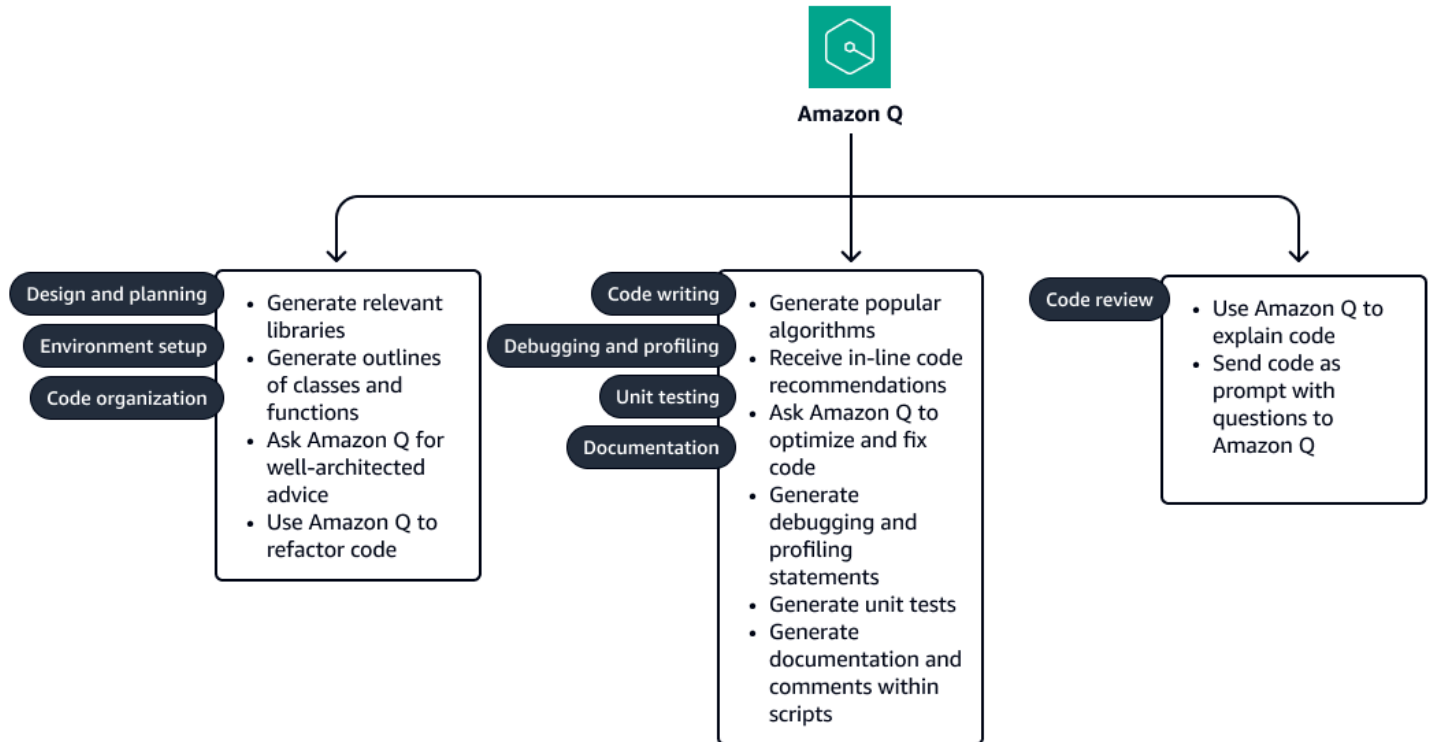
本指南為您提供以下有關使用 Amazon Q Developer 的洞察：

- 了解有效使用 Amazon Q Developer 進程式碼開發
 - 提供將 Amazon Q Developer 整合到[開發人員工作流程](#)的最佳實務。
 - 提供step-by-step指引，其中包含成功[產生程式碼](#)的範例和[建議](#)。
- 緩解常見挑戰並提升開發人員使用 Amazon Q Developer 的清晰度
 - 提供[策略](#)和洞見，以滿足開發人員的期望，並克服與程式碼產生準確性和效能相關的障礙。
- 提供疑難排解和錯誤處理

- 為開發人員提供 Amazon Q Developer 程式碼產生[疑難排解指引](#)，以解決不準確的結果或意外行為。
- 提供 [和 特定的實際範例和案例](#)Java。Python
- 最佳化工作流程和生產力
 - 使用 Amazon Q Developer 最佳化程式碼開發工作流程。
 - 討論提高[開發人員生產力](#)的策略。

在開發人員工作流程中使用 Amazon Q Developer

開發人員遵循標準工作流程，其中包含需求收集、[設計和規劃](#)、[編碼](#)、[測試](#)、[程式碼檢閱](#)和[部署](#)的階段。本節著重於如何使用 Amazon Q Developer 功能來最佳化金鑰開發步驟。



上圖顯示 Amazon Q 開發人員如何在程式碼開發階段加速和簡化下列常見任務：

- 設計與規劃 | 環境設定 | 程式碼組織
 - 產生相關程式庫
 - 產生類別和函數的概述
 - 向 Amazon Q 尋求架構良好的建議
 - 使用 Amazon Q 重構程式碼
- 程式碼編寫 | 除錯與分析 | 單位測試 | 文件
 - 產生常用演算法
 - 接收內嵌程式碼建議
 - 要求 Amazon Q 最佳化和修正程式碼
 - 產生偵錯和分析陳述式
 - 產生單位測試

- 在指令碼中產生文件和註解
- 程式碼檢閱
 - 要求 Amazon Q 解釋程式碼
 - 將程式碼當成提示傳送問題至 Amazon Q

設計和規劃

收集業務和技術需求後，開發人員會設計新的或擴展現有的程式碼庫。在此階段，Amazon Q Developer 可協助開發人員執行下列任務：

- 產生相關程式庫、類別和函數大綱，以取得架構良好的建議。
- 提供工程、相容性和架構設計查詢的指引。

編碼

編碼程序使用 Amazon Q 開發人員，以下列方式加速開發：

- 環境設定 - 在 AWS 工具組 整合式開發環境中安裝 (IDE) (例如 VS Code 或 IntelliJ)。然後，使用 Amazon Q 根據您的專案目標產生程式庫或接收設定建議。如需更多詳細資訊，請參閱 [Amazon Q 開發人員入門的最佳實務](#)。
- 程式碼組織 - 重構程式碼或從 Amazon Q 取得符合您專案目標的組織建議。
- 編寫程式碼 - 在開發時使用內嵌建議來產生程式碼，或要求 Amazon Q 使用 中的 Amazon Q 聊天面板來產生程式碼IDE。如需詳細資訊，請參閱[使用 Amazon Q Developer 產生程式碼的最佳實務](#)。
- 除錯與分析 - 產生分析命令，或使用 Fix and Explain 等 Amazon Q 選項來除錯問題。
- 單位測試 - 在聊天工作階段期間提供程式碼作為 Amazon Q 的提示，並請求產生適用的單位測試。如需詳細資訊，請參閱[使用 Amazon Q 開發人員 編寫程式碼範例](#)。
- 文件 - 使用內嵌建議來建立註解和文件字串，或使用說明選項來產生程式碼選擇的詳細摘要。如需詳細資訊，請參閱[使用 Amazon Q 開發人員 編寫程式碼範例](#)。

程式碼檢閱

檢閱者需要理解開發程式碼，才能將其提升至生產環境。若要加速此程序，請使用 Amazon Q 說明和最佳化選項，或在聊天工作階段中將程式碼選擇與自訂提示指示傳送至 Amazon Q。如需詳細資訊，請參閱[聊天範例](#)。

整合和部署

如需持續整合、交付管道和專案架構專用部署最佳實務的相關指引，請洽詢 Amazon Q。

使用這些建議，您可以學習有效地利用 Amazon Q 開發人員功能、最佳化工作流程，並提高整個開發生命週期的生產力。

Amazon Q Developer 的進階功能

雖然本指南著重於在實作程式設計任務中使用 Amazon Q Developer，但請務必注意其下列進階功能：

- Amazon Q 開發人員程式碼轉換
- Amazon Q 開發人員自訂

Amazon Q 開發人員程式碼轉換

適用於程式碼轉換的 Amazon Q 開發人員代理程式可以升級檔案的程式碼語言版本，而不需要手動重寫程式碼。它的運作方式是分析現有的程式碼檔案，並自動重寫這些檔案以使用較新版本的語言。例如，如果您在類似的 IDE 中工作，Amazon Q 會轉換單一模組 Eclipse。如果您使用的是 Visual Studio Code，Amazon Q 可以轉換整個專案或工作區。

當您想要執行常見的程式碼升級任務時，請使用 Amazon Q，如下所示：

- 更新程式碼以使用語言版本的新語法。
- 執行單元測試以驗證成功的編譯和執行。
- 檢查並解決部署問題。

Amazon Q 可以為開發人員節省數天到數個月的繁瑣重複工作，以升級程式碼庫。

自 2024 年 6 月起，Amazon Q Developer 支援升級 Java 程式碼，並可將 Java 8 個程式碼轉換為較新的版本，例如 Java 11 或 17。

Amazon Q 開發人員自訂

透過其自訂功能，Amazon Q Developer 可以根據公司自己的程式碼庫提供內嵌建議。公司將其程式碼儲存庫提供給 Amazon Simple Storage Service (Amazon S3) 或透過 AWS CodeConnections 提供，先前稱為 AWS CodeStar Connections。然後，Amazon Q 使用已啟用安全性的自訂程式碼儲存庫來建議與該組織中開發人員相關的編碼模式。

使用 Amazon Q Developer 自訂時，請注意下列事項：

- 截至 2024 年 6 月，Amazon Q 開發人員自訂功能處於預覽模式。因此，此功能在可用性和支援方面可能會受到限制。

- 自訂內嵌程式碼建議只會根據提供的程式碼儲存庫品質而準確。我們建議您檢閱所建立每個自訂項目的[評估分數](#)。
- 若要最佳化效能，建議您包含至少 20 個包含指定語言的資料檔案，其中所有來源檔案都大於 10MB。請確定您的儲存庫包含可參考的原始程式碼，而非中繼資料檔案（例如，組態檔案、屬性檔案和讀我檔案）。

透過使用 Amazon Q Developer 自訂，您可以透過以下方式節省時間：

- 使用以您自己的公司專屬程式碼為基礎的建議。
- 提高現有程式碼庫的重複使用性。
- 建立整個公司的可重複模式。

Amazon Q Developer 的最佳編碼實務

本節討論使用 Amazon Q Developer 進行編碼的最佳實務。最佳實務包括下列類別：

- [加入](#) - 加入時的方法和考量事項
- [程式碼產生](#) - 成功使用程式碼產生的指導
- [程式碼建議](#) - 改善程式碼的技術

加入 Amazon Q Developer 的最佳實務

Amazon Q Developer 是強大的生成式 AI 編碼助理，可透過 Visual Studio Code 和 JetBrains 等熱門 IDEs 取得。本節著重於存取 Amazon Q Developer 並將其加入您的編碼開發環境的最佳實務。

Amazon Q Developer 的先決條件

Amazon Q Developer 作為 AWS Toolkit for Visual Studio Code 和 的一部分提供 AWS Toolkit for JetBrains (例如，IntelliJ 和 PyCharm)。針對 Visual Studio Code 和 JetBrains IDEs，Amazon Q Developer 支援 Python、Java、JavaScript、TypeScript、C#、Go、Rust、PHP、Ruy、Kotlin、C、C++、Shell 指令碼、SQL 和 Scala。

如需 AWS 工具組 為 Visual Studio Code 和 JetBrains IDE 安裝 的詳細說明，請參閱 [《Amazon Q Developer 使用者指南》](#) 中的 [在 IDE 中安裝 Amazon Q Developer 延伸模組或外掛程式](#)。

使用 Amazon Q Developer 時的最佳實務

使用 Amazon Q Developer 的一般最佳實務包括下列項目：

- 提供相關內容以取得更準確的回應，例如程式設計語言、架構和正在使用的工具。將複雜的問題細分為較小的元件。
- 對您的提示和問題進行實驗和反覆運算。程式設計通常涉及嘗試不同的方法。
- 務必先檢閱程式碼建議，再接受它們，並視需要進行編輯，以確保它們完全按照您的預期執行。
- 使用 [自訂功能](#)，讓 Amazon Q Developer 了解您的內部程式庫、APIs、最佳實務和架構模式，以取得更相關的建議。

Amazon Q Developer 中的資料隱私權和內容使用量

決定使用 Amazon Q Developer 時，您應該了解如何使用您的資料和內容。以下是重點：

- 對於 Amazon Q Developer Pro 使用者，您的程式碼內容不會用於改善服務或模型訓練。
- 對於 Amazon Q Developer 免費方案使用者，您可以選擇不讓內容透過 IDE 設定或 AWS Organizations 政策用於改善服務。
- 傳輸的內容會加密，而任何儲存的內容都會以靜態加密和存取控制進行保護。如需詳細資訊，請參閱 [《Amazon Q 開發人員使用者指南》中的 Amazon Q 開發人員中的資料加密](#)。

使用 Amazon Q Developer 產生程式碼的最佳實務

Amazon Q Developer 提供自動程式碼產生、自動完成和自然語言程式碼建議。以下是使用 Amazon Q Developer 內嵌編碼協助的最佳實務：

- 提供內容以協助改善回應的準確性

從現有的程式碼開始，匯入程式庫，建立類別和函數，或建立程式碼骨架。此內容有助於大幅改善程式碼產生品質。

- 自然程式碼

像強大的自動完成引擎一樣使用 Amazon Q Developer 程式碼產生。像平常一樣編寫程式碼，並讓 Amazon Q 在您輸入或暫停時提供建議。如果無法使用程式碼產生，或遇到程式碼問題，請在 PC 上輸入 Alt+C 或 MacOS 上的 Option+C 來啟動 Amazon Q。如需使用內嵌建議時可採取之常見動作的詳細資訊，請參閱 [《Amazon Q 開發人員使用者指南》中的使用快速鍵](#)。

- 包含與指令碼目標相關的匯入程式庫

包含相關的匯入程式庫，以協助 Amazon Q 了解內容並相應地產生程式碼。您也可以要求 Amazon Q 建議相關的匯入陳述式。

- 維持清晰且集中的內容

讓您的指令碼專注於特定目標，並將不同的功能模組化為具有相關內容的個別指令碼。避免產生雜訊或混淆的內容。

- 使用提示進行實驗

探索不同的提示來微調 Amazon Q，以在程式碼產生過程中產生有用的結果。例如，嘗試以下方法：

- 針對自然語言提示使用標準註解區塊。

- 使用註解建立骨架，以填入類別和函數。
- 在您的提示中具體說明，提供詳細資訊而非一般化。
- 與 Amazon Q Developer 聊天並請求協助

如果 Amazon Q Developer 未提供準確的建議，請在 IDE 中與 Amazon Q Developer 聊天。它可以提供程式碼片段或完整類別和函數，以啟動您的內容。如需詳細資訊，請參閱 [《Amazon Q 開發人員使用者指南》](#) 中的 [與 Amazon Q 開發人員聊天有關程式碼](#)。

Amazon Q Developer 程式碼建議的最佳實務

Amazon Q Developer 可以提出開發人員問題並評估程式碼，以提供從程式碼產生和錯誤修正到使用自然語言指引的建議。以下是在 Amazon Q 中使用聊天的最佳實務：

- 從頭開始產生程式碼

對於新專案或當您需要一般函數時（例如，從 Amazon S3 複製檔案），請要求 Amazon Q Developer 使用自然語言提示產生程式碼範例。Amazon Q 可以提供公有資源的相關連結，以供進一步驗證和調查。

- 尋找編碼知識和錯誤說明

遇到編碼問題或錯誤訊息時，請提供程式碼區塊（如果適用，則提供錯誤訊息）和您的問題，做為 Amazon Q Developer 的提示。此內容將有助於 Amazon Q 提供準確且相關的回應。

- 改善現有的程式碼

若要修正已知錯誤或最佳化程式碼（例如，為了降低複雜性），請選取相關的程式碼區塊，並將其與您的請求一起傳送至 Amazon Q Developer。請具體說明您的提示，以獲得更好的結果。

- 說明程式碼功能

探索新的程式碼儲存庫時，請選取程式碼區塊或整個指令碼，並將其傳送給 Amazon Q Developer 進行說明。減少選取大小以取得更具體的說明。

- 產生單位測試

傳送程式碼區塊做為提示後，請要求 Amazon Q Developer 產生單元測試。這種方法可以節省程式碼涵蓋範圍和 DevOps 相關的時間和開發成本。

- 尋找 AWS 答案

Amazon Q Developer 是開發人員使用 AWS 服務的寶貴資源，因為它包含大量相關知識 AWS。無論您是面對特定的挑戰 AWS 服務、遇到特定錯誤訊息 AWS，或嘗試學習新的錯誤訊息 AWS 服務，Amazon Q 通常都會提供相關且有用的資訊。

請務必檢閱 Amazon Q Developer 提供給您的建議。然後，進行必要的編輯和執行測試，以確保程式碼符合您預期的功能。

Amazon Q Developer 的程式碼範例

本節提供逼真的範例，說明如何透過專注於 Python 和 Java 語言來改善您的體驗和程式碼產生。除了內嵌範例之外，某些案例顯示如何在程式設計時，使用 Amazon Q Developer 的聊天功能來請求程式碼建議的提示。

如需詳細資訊，請參閱下列主題：

- [Python 範例](#)
- [Java 範例](#)
- [聊天範例](#)

Python 範例

下列 Python 範例示範如何開始使用 Amazon Q Developer 產生內嵌程式碼建議。這些說明範例示範如何為您自己的專案觸發編碼助理。下列大多數範例適用於其他語言：

- [產生類別和函數](#)
- [文件程式碼](#)
- [產生演算法](#)
- [產生單位測試](#)

產生類別和函數

若要產生完整或部分類別實作，請使用程式碼註解。註解說明類別的意圖。開始簡單操作，並視需要新增更多詳細資訊。產生類別後，您可以繼續在其骨架內產生函數。

當您撰寫評論時，最好使用語言的評論結構。在 Python 中，對於單行註解，請使用 #。對於多行註解，請使用 ''' ''' 或 """ """。

輸入：

```
'''
DocumentReader class. has a name, children and optional parent.
Includes I/O functions to read and
write document
'''
```

輸出：

```
class DocumentReader:
    def __init__(self, name, children, parent=None):
        self.name = name
        self.parent = parent
        self.children = children

    def read(self):
        """Reads current document and returns text"""
        pass

    def write(self, text):
        """Writes text to current document"""
        pass
```

現在有了初始類別，可以使用額外的註解產生額外的 函數。

輸入：

```
'''
function to read document from local directory path and print the first line of the
document
'''
```

輸出：

```
def read_document(self, path):
    """Reads document from path and prints first line"""
    import os
    if os.path.isfile(path):
        with open(path, 'r') as f:
            print(f.readline())
    else:
        print(f"File not found at {path}")
```

文件程式碼

透過依賴內嵌文件來縮短程式碼文件中的時間。新增Python註解分隔符號 ''' '''。

輸入：

```
class S3Utils:
    ...
    <place cursor here>
    ...

    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
    print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

輸出：

```
class S3Utils:
    ...
    Class to generate S3 URL from bucket name and region
    ...

    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
    print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

產生演算法

Amazon Q Developer 可以產生熱門演算法，尤其是與 AWS SDKs 相關的演算法。在指令碼中，新增一個提示來指定您想要的函數。

此範例示範如何產生使用者 ID 索引鍵的字典來命名值，然後使用泡泡排序演算法產生排序函數。

輸入：

```
class SortList:
    @staticmethod
    def get_user_id_name_map():
        user_id_name_map = {
            1: "Amit",
            2: "Rahul",
            3: "Vijay",
            4: "Raj",
            5: "Kumar"
        }
        return user_id_name_map

...
Sort the list of user id's alphabetically using bubble sort
...
```

輸出：

```
@staticmethod
def sort_user_id_name_map():
    user_id_name_map = SortList.get_user_id_name_map()
    list_of_user_ids = list(user_id_name_map.keys())
    list_of_user_names = list(user_id_name_map.values())
    print("Before sort: ", list_of_user_ids)
    for i in range(len(list_of_user_ids)):
        for j in range(len(list_of_user_ids) - i - 1):
            if list_of_user_names[j] > list_of_user_names[j + 1]:
                list_of_user_ids[j], list_of_user_ids[j + 1] = list_of_user_ids[j +
1], list_of_user_ids[j]
                list_of_user_names[j], list_of_user_names[j + 1] =
list_of_user_names[j + 1], list_of_user_names[j]
    print("After sort: ", list_of_user_ids)
```

此外，您可以建立標準程序的提示。在此範例中，使用者會建立提示，將字典存放到 Amazon DynamoDB 資料表。在指令碼的開頭，使用者會新增提示來新增 DynamoDB 程式庫，以協助為 Amazon Q Developer 提供額外的內容。

輸入：

```
'''
Add libraries for Amazon Dynamo DB
'''
```

輸出：

```
'''
Add libraries for Amazon Dynamo DB and JSON.

import necessary libraries for working with Amazon DynamoDB and JSON data.
'''

import boto3
import json
from botocore.exceptions import ClientError

def load_data_to_dynamodb(table_name, data_file):
    # Create a DynamoDB resource
    dynamodb = boto3.resource('dynamodb')

    # Get a reference to the table
    table = dynamodb.Table(table_name)

    # Load data from the JSON file
    with open(data_file, 'r') as file:
        data = json.load(file)

    # Iterate over the data and load it to the table
    for item in data:
        try:
            table.put_item(Item=item)
            print(f"Successfully loaded data: {item}")
        except ClientError as e:
            print(f"Error loading data: {e.response['Error']['Message']}")

# Example usage
load_data_to_dynamodb('myTable', 'data.json')
```

現在，使用者可以輕鬆要求 Amazon Q 協助將字典儲存在 DynamoDB 中。

輸入：

```
'''
```

```
function to store the above sorted dict to dynamodb table named users
'''
```

輸出：

```
@staticmethod
def store_user_id_name_map_in_dynamodb():
    user_id_name_map = SortList.get_user_id_name_map()
    list_of_user_ids = list(user_id_name_map.keys())
    list_of_user_names = list(user_id_name_map.values())
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    for i in range(len(list_of_user_ids)):
        try:
            response = table.put_item(
                Item={
                    'user_id': list_of_user_ids[i],
                    'user_name': list_of_user_names[i]
                }
            )
        except ClientError as e:
            print(e.response['Error']['Message'])
```

產生單位測試

透過內嵌註解，Amazon Q Developer 可以產生單元測試，以執行聲明和其他測試。在此範例中，使用者要求 Amazon Q 產生適用的程式庫以進行單元測試，以設定測試指令碼的內容。然後，使用者要求 Amazon Q 為 `sort_user_id_name_map` 函數撰寫測試函數。

輸入：

```
'''
Import all the required unit test libraries for python and then generate a unit test
for sort_user_id_name_map function.
'''
```

輸出：

```
import unittest
from unittest.mock import patch
from sort_user_id_name_map import sort_user_id_name_map
```

```
class TestSortUserIdNameMap(unittest.TestCase):
    @patch('sort_user_id_name_map.sort_user_id_name_map')
    def test_sort_user_id_name_map(self, mock_sort):
        mock_sort.return_value = {'id1': 'name1', 'id2': 'name2'}
        result = sort_user_id_name_map({'id2': 'name2', 'id1': 'name1'})
        self.assertEqual(result, {'id1': 'name1', 'id2': 'name2'})

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

此外，在使用者和 Amazon Q Developer 之間的聊天中，它會根據使用者的程式碼輸入提示來建議和產生單元測試。如需詳細資訊，請參閱[聊天範例](#)。

Java 範例

下列Java範例示範使用者如何開始產生內嵌程式碼建議。這些說明範例示範如何為您自己的專案觸發編碼助理。下列大多數範例適用於其他語言：

- [產生類別和函數](#)
- [文件代碼](#)
- [產生演算法](#)
- [產生單位測試](#)

產生類別和函數

若要產生完整或部分類別實作，請使用程式碼註解。註解說明類別的意圖。開始簡單操作，並視需要新增更多詳細資訊。產生類別後，您可以繼續在其骨架內產生函數。

撰寫評論時，最好使用語言的評論結構。在中Java，這是 `/** */`。

輸入：

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
```

輸出：

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
import java.util.ArrayList;
public class DocumentReader {
    private String name;
    private DocumentReader parent;
    private ArrayList<DocumentReader> children;

    /**
     * Constructor
     * @param name name of the document
     * @param parent parent of the document
     */
    public DocumentReader(String name, DocumentReader parent) {
        this.name = name;
        this.parent = parent;
        this.children = new ArrayList<>();
    }

    /**
     * Constructor
     * @param name name of the document
     */
    public DocumentReader(String name) {
        this(name, null);
    }

    /**
     * Getter for name
     * @return name of the document
     */
    public String getName() {
        return name;
    }

    /**
     * Getter for parent
     * @return parent of the document
     */
    public DocumentReader getParent() {
```

```
        return parent;
    }

    /**
     * Getter for children
     * @return children of the document
     */
    public ArrayList<DocumentReader> getChildren() {
        return children;
    }

    /**
     * Adds a child to the document
     * @param child child to be added
     */
    public void addChild(DocumentReader child) {
        children.add(child);
    }

    /**
     * Removes a child from the document
     * @param child child to be removed
     */
    public void removeChild(DocumentReader child) {
        children.remove(child);
    }
}
```

現在有了初始類別，可以使用額外的註解產生額外的 函數。

輸入:

```
/**
 * function to read document from local directory path
 * and print first line of document
 */
```

輸出 :

```
/**
 * function to read document from local directory path
 * and print first line of document
 */
```

```
public void readDocument() {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(name));
        String line = reader.readLine();
        System.out.println(line);
        reader.close();
    } catch (IOException e) {
        System.out.println("Error reading document: " + e.getMessage());
    }
}
```

文件程式碼

透過依賴內嵌文件來縮短程式碼文件中的時間。新增Java評論休息時間 `/* */`。

輸入:

```
import java.util.function.Function;

public class S3Utils {
    /*
     * <place cursor here>
     */
    public static Function<String, String> generateS3Function(String bucketName, String
region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

如果未發生內嵌產生，請在將游標放在您想要文件的位置之後執行 Amazon Q Developer 內嵌建議。

輸出：

```
import java.util.function.Function;

public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
     */
}
```

```
    * @param region the region of the bucket
    * @return the function that generates the S3 URL
    */
    public static Function<String, String> generateS3Function(String bucketName, String
region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

如需更簡單的程式碼說明（類別、函數或指令碼元素），請使用策略上放在您要新增文件之程式碼區段前面的轉送字元（//）。

輸入：

```
//
public class S3Utils {
    /*
    * Generate a function that returns the S3 URL for a given key
    * @param bucketName the name of the bucket
    * @param region the region of the bucket
    * @return the function that generates the S3 URL
    */
    public static Function<String, String> generateS3Function(String bucketName, String
region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

輸出：

```
// This class is used to generate a function that returns the S3 URL for a given key
public class S3Utils {
    /*
    * Generate a function that returns the S3 URL for a given key
    * @param bucketName the name of the bucket
    * @param region the region of the bucket
    * @return the function that generates the S3 URL
```

```
    */
    public static Function<String, String> generateS3Function(String bucketName, String
region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

產生演算法

Amazon Q Developer 可以產生熱門演算法，尤其是與 AWS SDKs 相關的演算法。在指令碼中，新增一個提示來指定您想要的函數。

此範例會產生使用者 ID 索引鍵的雜湊圖來命名值，然後使用泡泡排序演算法產生排序函數。

輸入：

```
import java.util.HashMap;
public class SortList {

    // generate key value pair hash map for
    // user id and name
    public static HashMap<Integer, String> getUserIdNameMap() {
        HashMap<Integer, String> userIdNameMap = new HashMap<Integer, String>();
        userIdNameMap.put(1, "Amit");
        userIdNameMap.put(2, "Rahul");
        userIdNameMap.put(3, "Vijay");
        userIdNameMap.put(4, "Raj");
        userIdNameMap.put(5, "Kumar");
        return userIdNameMap;
    }

    /*
    * Create function to sort UserIdName map by name alphabetically using bubble sort
    */
}
```

輸出：

```
/*
 * Create function to sort UserIdName map by name alphabetically using bubble sort
 */
public static HashMap<Integer, String> sortUserIdNameMap(HashMap<Integer, String>
  userIdNameMap) {
    int size = userIdNameMap.size();
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (userIdNameMap.get(i).compareTo(userIdNameMap.get(j)) > 0) {
                String temp = userIdNameMap.get(i);
                userIdNameMap.put(i, userIdNameMap.get(j));
                userIdNameMap.put(j, temp);
            }
        }
    }
    return userIdNameMap;
}
```

此外，使用者可以建立標準程序的提示。在此範例中，使用者會建立提示，將雜湊映射儲存到 Amazon DynamoDB 資料表。在指令碼的開頭，使用者會新增提示來新增 DynamoDB Java 程式庫，以協助為 Amazon Q Developer 提供額外的內容。

輸入:

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

/*
 * Store sorted UserIdName map into Amazon DynamoDB table named users
 */
```

輸出:

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

/*
 * Store sorted UserIdName map into Amazon DynamoDB table named users
 */
```

```
public static void storeSortedUserIdNameMap(HashMap<Integer, String> userIdNameMap)
{
    // create Amazon DynamoDB client
    AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();

    // store sorted UserIdName map into Amazon DynamoDB table named users
    for (int userId : userIdNameMap.keySet()) {
        String name = userIdNameMap.get(userId);
        client.putItem("users", new HashMap<String, Object>() {{
            put("userId", userId);
            put("name", name);
        }});
    }
}
```

產生單位測試

透過內嵌註解，Amazon Q Developer 可以產生單元測試，以執行聲明和其他 JUnit 測試。在此範例中，使用者會先要求 Amazon Q 為 JUnit 產生適用的程式庫，以設定測試指令碼的內容。接著，使用者會建立公有類別，後面接著宣告者和評論字串。

輸入：

```
// import libraries for java unit tests
import org.junit.Test;
import static org.junit.Assert.*;

public class UnitTests {

    @Test
    /*
     * Create unit test for getUserIdNameMap
     */
}
```

輸出：

```
@Test
/*
 * Create unit test for getUserIdNameMap
 */
public void testGetUserIdNameMap() {
```

```
// create a new instance of the UserIdNameMap class
UserIdNameMap userIdNameMap = new UserIdNameMap();

// create a new instance of the User class
User user = new User("John", "Doe");

// add the user to the map
userIdNameMap.addUser(user);

// get the user from the map
User retrievedUser = userIdNameMap.getUser("John");

// assert that the retrieved user is the same as the user added to the map
assertEquals(user, retrievedUser);
}
```

此外，在使用者和 Amazon Q Developer 之間的聊天中，它會根據使用者的程式碼輸入提示來建議和產生單元測試。如需詳細資訊，請參閱[聊天範例](#)。

聊天範例

Amazon Q 開發人員聊天可協助開發人員進行各種任務，包括編碼、測試和升級應用程式、診斷錯誤、執行安全性掃描和修正，以及最佳化 AWS 資源。本節說明與 Amazon Q 聊天如何在一般程式設計任務中支援開發人員，例如：

- [詢問 AWS 服務](#)
- [產生程式碼](#)
- [產生單位測試](#)
- [說明程式碼](#)

詢問 AWS 服務

使用者可以詢問 Amazon Q Developer 有關的問題 AWS 服務，例如尋找正確的服務或了解最佳實務。

有關的聊天 AWS 服務

使用者：

如果我需要降低成本，最適合用來執行工作負載的容器服務是什麼？

Amazon Q :

根據提供的資訊，您似乎正在尋找經濟實惠的容器服務來執行工作負載。在這種情況下，我會建議考慮使用 Fargate 來使用 Amazon Elastic Container Service (Amazon ECS)。[1]

Amazon ECS with Fargate 是容器的無伺服器運算引擎，可讓您執行容器，而無需管理任何基礎設施。使用 Fargate，您不需要佈建或管理伺服器，Fargate 會為您處理。這有助於降低成本，因為您只需要支付容器實際使用的運算資源，而不必支付閒置伺服器容量的費用。【2】

搭配 Fargate 使用 Amazon ECS 的一些主要優點：

- 不需要管理伺服器或叢集 – Fargate 會為您處理基礎設施佈建和管理。
- Pay-as-you-go定價 – 您只需為容器使用的運算資源付費，而不必為閒置容量付費。
- 自動擴展 – Fargate 可以根據需求自動擴展和縮減您的容器。
- 與其他 緊密整合，AWS 服務 例如負載平衡、記錄和監控。

所提供的資源也提到 ECS 服務在容器排程和置放方面提供靈活性，這有助於進一步最佳化您的成本。

建議您在 AWS 網站上檢閱 Amazon ECS 搭配 Fargate 的定價詳細資訊，以根據您的特定工作負載需求更清楚成本。當然，如果您有任何其他問題，請隨時通知我！

來源

【1】 哪種 AWS 容器服務最適合您？

aptgetops.tech > [aws-container-services-which-one-is-right-for-you](#)

【2】 非 HTTP 工作負載的Java容器選項

https : // > 問題 > [43417016](#)

結束有關的 Amazon Q 聊天 AWS 服務

產生程式碼

使用者可以要求 Amazon Q Developer 協助撰寫程式碼片段。

有關編寫程式碼的聊天

使用者：

是否可以提供如何使用 列出所有 S3 儲存貯體的範例 適用於 Python (Boto) 的 AWS SDK ?

Amazon Q :

當然，以下範例說明如何使用 適用於 Python (Boto) 的 AWS SDK **【1】** 列出所有 S3 儲存貯體：

```
import boto3
# Create an S3 client
s3 = boto3.client('s3')

# List all the buckets
response = s3.list_buckets()

# Print the name of each bucket
for bucket in response['Buckets']:
    print(bucket['Name'])
```

以下是程式碼的運作方式：

1. 我們首先匯入 boto3 程式庫，這是適用於 Python 的 SDK。
2. 我們會使用 建立 S3 用戶端 `boto3.client('s3')`。
3. 我們在 S3 用戶端上呼叫 `list_buckets()` 方法，傳回包含帳戶中所有儲存貯體清單的回應。
4. 然後，我們會逐一查看回應中的 Buckets 清單，並列印每個儲存貯體的名稱。

這是基本範例，但 適用於 Python (Boto) 的 AWS SDK 提供許多更進階的功能來使用 Amazon S3，例如建立、刪除和管理儲存貯體中的物件、設定儲存貯體政策等。您可以在 適用於 Python (Boto) 的 AWS SDK) 文件中找到更多範例和文件。 **【2】**

請記住，使用時 AWS 服務，請務必遵循處理登入資料和敏感資訊的最佳實務，例如將存取金鑰存放在 AWS 環境變數中，而不是在程式碼中硬式編碼。

來源：**【1】** [使用 SDK for Python \(Boto3\) 的 Amazon S3 範例 - AWS SDK 程式碼範例 docs.aws.amazon.com > code-library > latest-1x > ... > python-s3.html](https://docs.aws.amazon.com/code-library/latest-1x/python-s3.html)

【2】 [使用 AWS SDK - AWS SDK 程式碼範例開始使用 Amazon S3 儲存貯體和物件 docs.aws.amazon.com > code-library > latest-1x > ... > s3_example_s3_buckets_GettingStarted_sect...](https://docs.aws.amazon.com/code-library/latest-1x/s3_example_s3_buckets_GettingStarted_sect...)

結束有關編寫程式碼的 Amazon Q 聊天

產生單位測試

使用者可以將選取的程式碼做為提示傳送給 Amazon Q Developer，其中會套用其他指示，例如產生單位測試。如需step-by-step說明，請參閱 [《Amazon Q 開發人員使用者指南》](#) 中的將程式碼傳送至 Amazon Q。

單元測試相關聊天

使用者：

使用模擬資料撰寫下列函數的單元測試：

```
// poll messages from Amazon SQS queue name
public List<String> pollMessages(String queueName) {
    List<String> messages = new ArrayList<>();
    // poll messages from queue
    ReceiveMessageResult = sqsClient.receiveMessage(new ReceiveMessageRequest().
    for (Message : receiveMessageResult.getMessages()) {
        messages.add(message.getBody());
    }
    return messages;
}
```

Amazon Q：

若要使用 pollMessages 函數的模擬資料撰寫單位測試，我們可以使用模擬架構，例如 Mockito。範例如下：

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
@ExtendWith(MockitoExtension.class)
public class SQSUtilsTest {

    @Mock
    private AmazonSQS sqsClient;

    @InjectMocks
    private SQSUtils sqsUtils;

    @Test
    public void testPollMessages() {
        // Arrange
        String queueName = "test-queue";
        Message message1 = new Message().withBody("message1");
        Message message2 = new Message().withBody("message2");
        ReceiveMessageResult receiveMessageResult = new
ReceiveMessageResult().withMessages(message1, message2);
        Mockito.when(sqsClient.receiveMessage(new
ReceiveMessageRequest().withQueueUrl(queueName)))
                .thenReturn(receiveMessageResult);

        // Act
        List<String> messages = sqsUtils.pollMessages(queueName);

        // Assert
        assertEquals(Arrays.asList("message1", "message2"), messages);
    }
}
```

在此範例中，我們使用 Mockito 來模擬 AmazonSQS 用戶端，並設定 `receiveMessage` 方法的預期行為，以便傳回兩個模擬訊息，其中包含「message1」和「message2」內文。然後，我們呼叫 `pollMessages` 方法，並宣告傳回的訊息清單符合預期值。

關鍵點為：

1. 我們使用 `@ExtendWith(MockitoExtension.class)` 註釋來啟用 Mockito 與 JUnit 的整合。
2. 我們使用 `@Mock` 註釋來建立 AmazonSQS 用戶端的模擬執行個體。
3. 我們使用 `@InjectMocks` 註釋將模擬 AmazonSQS 用戶端插入 `SQSUtils` 類別。
4. 在測試方法中，我們使用 `Mockito.when().thenReturn()` 設定 `receiveMessage` 方法的預期行為。
5. 我們呼叫 `pollMessages` 方法，並宣告傳回的訊息清單符合預期值。

這種方法可讓我們在不實際與 Amazon SQS 服務互動的情況下測試 `pollMessages` 方法，這可讓測試更快、更可靠。

結束有關單元測試的 Amazon Q 聊天

說明程式碼

對於程式碼檢閱程序，開發人員可以將程式碼選取項目傳送至 Amazon Q Developer，以進一步了解程式碼。此外，Amazon Q 提供的說明也可以用於文件目的。在下列範例中，Amazon Q 會使用 函數說明處理常式 AWS Lambda 函數，然後提出 Docstring。

有關解釋程式碼的聊天

使用者：

說明我程式碼的下列部分：

```
def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    s3_client = boto3.client('s3')
    table = dynamodb.Table('DYNAMODB_TABLE')
    # Get s3 object from the event
    print(event)
    print(context)
    # get object from event
    bucket_key = event['Records'][0]['s3']['object']['key']
    object_key = unquote_plus(bucket_key)
    fullText = detect_text(object_key, bucket)

    response = s3_client.head_object(Bucket=bucket, Key=object_key)
    print(response)
    print(fullText)
    metadata = response['ResponseMetadata']
    metadata = json.dumps(metadata)

    now = datetime.now()
    current_time = now.strftime("%A %d %B, %Y %H:%M:%S")
    print("Current Time =", current_time + "\n\n")
    reason, outcome = detect_restaurant_closure(fullText, current_time)
    table.put_item(
        Item={
            'id': object_key,
            'creationTime': now.strftime("%Y-%m-%d %H:%M:%S"),
```

```
        'reason': reason,\n        'text': fullText,\n        'metadata': metadata\n    }\n)
```

Amazon Q :

提供的程式碼區塊似乎是寫入的 AWS Lambda 函數 Python。讓我解釋此程式碼的關鍵層面：

1. 函數名為 `lambda_handler`，是 Lambda 函數的進入點。
2. 它使用 `boto3` 程式庫與 AWS DynamoDB 和 S3 服務互動。
3. 函數會從傳入的事件資料擷取 S3 物件金鑰和儲存貯體名稱。
4. 然後，它會呼叫 `detect_text` 函數（未顯示在提供的程式碼中）來擷取 S3 物件的完整文字內容。
5. 函數也會使用 `head_object` 方法擷取 S3 物件的中繼資料。
6. 然後，它會擷取目前的時間，並呼叫 `detect_restaurant_closure` 函數（也未顯示），根據擷取的文字來判斷餐廳關閉的原因和結果。
7. 最後，函數會將下列資訊存放在 DynamoDB 資料表中：
 - 做為 ID 的 S3 物件金鑰
 - 建立時間
 - 餐廳關閉的原因和結果
 - 擷取的 S3 物件文字

整體而言，此 Lambda 函數似乎是處理 S3 物件、擷取文字內容、分析餐廳關閉資訊，以及將結果存放在 DynamoDB 資料表中的大型系統的一部分。需要 `detect_text` 和 `detect_restaurant_closure` 函數的特定詳細資訊，才能完全了解此程式碼的功能。

Amazon Q 回應結束

使用者：將此說明摘要為此 `lambda_handler` 函數的 Docstring

Amazon Q：這是 `lambda_handler` 函數的建議 Docstring **【1】**：

```
def lambda_handler(event, context):\n    \"\"\"\n        AWS Lambda function that processes an S3 object, extracts text content, analyzes\n        it, and stores the results.\n        Args:\n    \"\"\"
```

```
    event (dict): The event data that triggered the Lambda function, containing
information about the S3 object.
    context (object): The context object provided by AWS Lambda, containing
information about the runtime environment.

Returns:
    None
"""

# function implementation details...
```

此 docstring 提供函數用途的高階摘要、預期的輸入參數，以及預期的傳回值（或缺少此值）。它可讓使用者清楚了解 `lambda_handler` 函數的功能，而無需深入了解實作詳細資訊。

來源：【1】 在 C# - AWS Lambda <https://docs.aws.amazon.com/lambda/latest/dg/csharp-handler.html> 中定義 Lambda 函數處理常式 - :~: text=Lambda 函數處理常式為、結束或逾時。

有關解釋程式碼的 Amazon Q 聊天結束

對 Amazon Q Developer 中的程式碼產生案例進行故障診斷

使用 Amazon Q Developer 時，您可能會遇到下列程式碼產生和解析度不正確的常見案例：

- [產生空的程式碼](#)
- [持續註解](#)
- [不正確的內嵌程式碼產生](#)
- [聊天結果不足](#)

產生空的程式碼

開發程式碼時，您可能會注意到下列問題：

- Amazon Q 不提供建議。
- 訊息「無來自 Amazon Q 的建議」會顯示在您的 IDE 中。

您的第一個想法可能是 Amazon Q 無法正常運作。不過，這些問題的根本原因通常與指令碼中的內容或在 IDE 中開啟的專案相關聯。

如果 Amazon Q Developer 未自動提供建議，您可以使用下列捷徑手動執行 Amazon Q 內嵌建議：

- PC - Alt+C
- MacOS - Option+C

如需詳細資訊，請參閱《Amazon Q 開發人員使用者指南》中的[使用快速鍵](#)。

在大多數情況下，Amazon Q 會產生建議。當 Amazon Q 傳回「無來自 Amazon Q 的建議」訊息時，請檢閱此問題的下列常見解決方案：

- 內容不足 – 確定存在適用的程式庫。確定指令碼中定義的類別和函數與新程式碼相關。
- 模稜兩可的提示 – 如果提示混淆或不明確，Amazon Q 會在提出內嵌程式碼建議時表現不佳。在這些案例中，請在程式碼和註解中更具體地引出適當的回應。

如果您停滯，請盡可能繼續編碼。一般而言，Amazon Q 會辨識您的意圖，這會觸發它提供建議。另一個選項是傳送程式碼片段，其中包含 Amazon Q 聊天中的提示說明。如需詳細資訊，請參閱《[Amazon Q 開發人員使用者指南](#)》中的[將程式碼傳送至 Amazon Q](#)。

持續註解

您可以使用註解撰寫提示，而且只會從 Amazon Q 接收一般或重複的註解行。

輸入：

```
// Kafka test
```

結果：

```
// Kafka test
// Kafka test
// Kafka test
// Kafka test
```

一般而言，此問題是由於指令碼缺乏內容或提示指示不佳所致。我們建議您將標準格式用於指定語言的註解。在此範例中，使用者變更了用於註解區塊 `/* */` 的提示，該變更從 Amazon Q 獲得更好的效能。

輸入：

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
```

結果：

在此結果中，使用者依序接受數個建議。

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
import org.junit.Test;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import static org.junit.Assert.assertEquals;
// create function for a Kafka test
public class KafkaTest {
    @Test
```

```
public void testKafka() {
    // create Kafka producer
    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092");
    props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
    KafkaProducer<String, String> producer = new KafkaProducer<>(props);
    // create Kafka topic
    String topic = "test";
    // send message to Kafka topic
    producer.send(new ProducerRecord<>(topic, "key", "value"));
    // close Kafka producer
    producer.close();
    // assert that message was sent successfully
    assertEquals("value", KafkaTest.consumer.poll().value());
}
}
```

不正確的內嵌程式碼產生

有時，Amazon Q Developer 內嵌建議沒有用或不符合提示的目標。有時，內嵌程式碼建議可能會提供類似簡單 Hello World 方法的一般函數。

當內嵌程式碼建議沒有幫助時，請嘗試下列最佳實務：

- 檢閱指令碼的內容。通常，您提供的內容越多，您從 Amazon Q 獲得的結果就越好。
- 如果內容不同和/或吵雜，請將您的程式碼模組化。將相同指令碼中的類似函數分組。依目標分隔不同的類別和函數。
- 以策略方式撰寫您的類別和函數名稱，以協助 Amazon Q 提供更好的建議。

在此範例中，開發人員從頭開始撰寫指令碼，將檔案上傳至 Hadoop。

輸入：

```
class MyFunction{

    // create function to read data locally and upload to HDFS filesystem
```

結果：

```
class MyFunction{

    // create function to read local data file path and upload to HDFS filesystem
    public void readAndUploadToHDFS(String inputPath, String outputPath) throws
    IOException {
        // read data from local file
        BufferedReader in = new BufferedReader(new FileReader(inputPath));
    }
}
```

雖然 Amazon Q 提供建議，但不符合開發人員的期望。為了改善效能，開發人員會匯入程式庫，並將類別名稱變更為代表 Hadoop 公用程式。

修正：

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;

// import applicable hadoop libraries for IO operations
import org.apache.hadoop.fs.Path;

class HadoopUtilities{

    // create function to read local data file path and upload to HDFS filesystem
}
```

結果：

在此結果中，開發人員會依序接受數個建議。

```
class HadoopUtilities{

    // create function to read local data file path and upload to HDFS filesystem
    public static void uploadToHDFS(String localFilePath, String hdfsFilePath) throws
    IOException {
        // create HDFS file path object
        Path hdfsPath = new Path(hdfsFilePath);
        // create local file path object
        Path localPath = new Path(localFilePath);
    }
}
```

```
// create hadoop file system object
org.apache.hadoop.fs.FileSystem hdfs = org.apache.hadoop.fs.FileSystem.get(new
org.apache.hadoop.conf.Configuration());
// upload local file to HDFS
hdfs.copyFromLocalFile(localPath, hdfsPath);
// close HDFS file system
hdfs.close();
System.out.println("File uploaded successfully!");
}
```

在相同的HadoopUtilities指令碼中，開發人員新增註解提示來讀取 Amazon Simple Queue Service (Amazon SQS) 主題。

輸入:

```
// read message from SQS
```

結果：

```
// read message from SQS
public static void readMessageFromSQS(String message) {
    System.out.println("Received message from SQS: " + message);
}
```

與其使用 Java 方法從 Amazon SQS 讀取 SQS 訊息，Amazon Q 建議列印字串方法。不過，此建議不符合開發人員的原始目標。

在這種情況下，最好重新修改程式碼。指令碼的重點是 Hadoop 方法。因此，建立專門用於 SQS 操作的公用程式指令碼可能是更好的方法。

為 SQS 公用程式建立個別類別可改善內嵌建議，此建議可用於HadoopUtilities指令碼。

輸入:

```
// import function for SQS
import com.amazonaws.services.sqs.AmazonSQS;
import java.util.ArrayList;
import java.util.List;

class SQSUtils {
```

```
// instantiate SQS client
private AmazonSQS sqsClient;

// load SQS object
public SQSUtils(AmazonSQS sqsClient) {
    this.sqsClient = sqsClient;
}

// poll message from Amazon SQS given queue name
}
```

結果：

```
// poll message from Amazon SQS given queue name
public List<String> pollMessages(String queueName) {
    List<String> messages = new ArrayList<>();
    // poll messages from queue
    receiveMessageResult = sqsClient.receiveMessage(new
ReceiveMessageRequest(queueName));
    for (Message message : receiveMessageResult.getMessages()) {
        messages.add(message.getBody());
    }
    return messages;
}
```

如果內嵌建議仍然未提供預期結果，請在聊天工作階段中要求 Amazon Q Developer 產生程式碼。

有關產生程式碼的聊天

使用者：

產生 Java 函數以從 Amazon SQS 佇列讀取訊息

Amazon Q：

以下是從 Amazon SQS 佇列讀取訊息的範例 Java 函數：

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
public class SQSMessageReader {
```

```
private AmazonSQS sqsClient;
private String queueUrl;

public SQSMessageReader(AmazonSQS sqsClient, String queueUrl) {
    this.sqsClient = sqsClient;
    this.queueUrl = queueUrl;
}

public String readMessage() {
    ReceiveMessageRequest receiveMessageRequest = new
ReceiveMessageRequest(queueUrl)
        .withMaxNumberOfMessages(1)
        .withWaitTimeSeconds(10);

    ReceiveMessageResult receiveMessageResult =
sqsClient.receiveMessage(receiveMessageRequest);

    if (receiveMessageResult.getMessages().size() > 0) {
        Message message = receiveMessageResult.getMessages().get(0);
        String messageBody = message.getBody();
        String receiptHandle = message.getReceiptHandle();

        // Process the message
        System.out.println("Received message: " + messageBody);

        // Delete the message from the queue
        sqsClient.deleteMessage(queueUrl, receiptHandle);

        return messageBody;
    } else {
        return null;
    }
}
```

此程式碼會建立採用 AmazonSQS 用戶端和佇列 URL 做為建構函數參數的 SQSMessageReader 類別。readMessage() 方法使用 ReceiveMessageRequest 從佇列擷取最多 1 則訊息，最長等待時間為 10 秒。

如果收到訊息，則會列印訊息內文，並使用 DeleteMessageRequest 從佇列中刪除訊息。接著會傳回訊息內文。如果未收到任何訊息，則會傳回 null。

結束有關產生程式碼的 Amazon Q 聊天

聊天結果不足

在開發程式碼時，開發人員可以要求 Amazon Q 產生常見函數、提供建議或解釋程式碼。有時，如果出現提示（例如問題或程式碼片段），Amazon Q 可能會提供一般意見回饋或不符合您期望的結果。在這些案例中，請嘗試下列動作：

- 使用不同的提示進行實驗，以改善您從 Amazon Q 取得的程式碼產生結果。
- 遵循 Amazon Q 在回應您的提示或問題時經常提供的 URLs。
- 提供意見回饋以協助改善 Amazon Q 的未來效能。

例如，開發人員正在建立擷取增強生成 (RAG) 解決方案，他們需要根據句子之間的語意關係來區塊其文件。開發人員與 Amazon Q 聊天，如下所示。

討論 RAG 解決方案

使用者：

對於 RAG 解決方案，產生語意區塊化的程式碼

Amazon Q：

當然，以下範例說明如何 Python 使用 LangChain 程式庫在中實作擷取增強生成 (RAG) 解決方案的語意區塊：

```
from langchain.text_splitter import CharacterTextSplitter, TokenTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = TokenTextSplitter(
    chunk_size=500,
    chunk_overlap=50,
    length_function=len
)
# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
```

```
)  
  
# Chunk the text using semantic similarity  
def semantic_chunk(text):  
    chunks = text_splitter.split_text(text)  
    embeddings = embeddings.embed_documents(chunks)  
    vectorstore.add_texts(chunks, embeddings)  
    return chunks
```

在此範例中，我們使用 `TokenTextSplitter` 來自 `LangChain` 的，將輸入文字分割為 500 個字元區塊，並重疊 50 個字元。此 `HuggingFaceEmbeddings` 模型用於為每個區塊產生內嵌，而 `Chroma` 向量存放區用於存放區塊及其內嵌。

`semantic_chunk` 函數採用輸入文字，使用 `text_splitter` 將其分割為區塊，使用內嵌模型為每個區塊產生內嵌，然後將區塊及其內嵌新增至 `vectorstore`。

此方法可讓您將輸入文字分割為語意有意義的區塊，這對於 RAG 解決方案中的問題回答或摘要等各種任務很有用。

請注意，這是基本範例，您可能需要根據您的特定使用案例和資料來調整參數（例如 `chunk_size`、`chunk_overlap`）或文字分割器和內嵌模型的選擇。

有關 RAG 解決方案的 Amazon Q 聊天結束

不過，此方法只提供使用滑動視窗技術來區塊文件的基本方法。開發人員想要一種依賴於測量內嵌的進階方式。在這些案例中，開發人員必須根據其目標，將更多詳細資訊新增至提示。當開發人員參考 `LangChain` 的已知 AI21 文字分割器函數時，他們會收到改善的回應，如下列聊天所示。

討論 RAG 解決方案（有改善的提示）

使用者：

上述提示僅與每個區塊的字符大小分割相關。我想要使用 AI21 語意文字分割器。使用具有 `langchain` 的 AI21 語意文字分割來產生函數。

Amazon Q：

好的，讓我們使用 AI21 語意文字分割器搭配 `LangChain` 來實作語意區塊。以下是範例：**【1】**

```
from langchain.text_splitter import AI21SemanticTextSplitter  
from langchain.embeddings import HuggingFaceEmbeddings
```

```
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = AI21SemanticTextSplitter()

# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
)

# Chunk the text using semantic similarity
def semantic_chunk(text):
    chunks = text_splitter.split_text(text)
    embeddings = embeddings.embed_documents(chunks)
    vectorstore.add_texts(chunks, embeddings)
    return chunks
```

在此範例中，我們使用來自 LangChain 的 AI21SemanticTextSplitter，將輸入文字分割為語意有意義的區塊。此 HuggingFaceEmbeddings 模型用於為每個區塊建立內嵌，而 Chroma 向量存放區用於存放區塊及其內嵌。

semantic_chunk 函數採用輸入文字，使用 text_splitter 將其分割為區塊，使用內嵌模型為每個區塊建立內嵌，然後將區塊及其內嵌新增至 vectorstore。

此方法可讓您將輸入文字分割為語意有意義的區塊，這對於 RAG 解決方案中的問題回答或摘要等各種任務很有用。

請注意，您可能需要根據您的特定使用案例和資料來調整參數或內嵌模型的選擇。

來源 **【1】** 了解如何在 LangChain 中使用文字分割器

[soosweb3.hashnode.dev > the-ultimate-langchain-series-text-splitters](https://soosweb3.hashnode.dev/the-ultimate-langchain-series-text-splitters)

有關 RAG 解決方案的 Amazon Q 聊天結束（有改善的提示）

根據聊天結果，開發人員可以使用函數並導覽至資訊來源。

一般而言，當您與 Amazon Q Developer 聊天時，進階主題和問題需要在提示中更具體的內容。如果您認為聊天的結果不正確，請使用拇指向下圖示提供有關 Amazon Q 回應的意見回饋。Amazon Q

Developer 會持續使用意見回饋來改善未來的版本。對於產生正面結果的互動，使用大拇指圖示提供意見回饋很有用。

Amazon Q Developer 的FAQs

本節提供有關使用 Amazon Q Developer 進行程式碼開發的常見問題解答。

什麼是 Amazon Q Developer ？

Amazon Q Developer 是一項強大的生成式 AI 驅動服務，旨在透過提供智慧程式碼產生和建議來加速程式碼開發任務。在 2024 年 4 月 30 日，Amazon CodeWhisperer 成為 Amazon Q Developer 的一部分。

如何存取 Amazon Q Developer ？

Amazon Q Developer 可作為 AWS Toolkits for Visual Studio Code 和 JetBrains IDEs 的一部分提供，例如 IntelliJ 和 PyCharm。若要開始使用，[請安裝 AWS 工具組 最新版本](#)。

Amazon Q Developer 支援哪些程式設計語言 ？

針對 Visual Studio 程式碼和 JetBrains IDEs，Amazon Q Developer 支援 Python、Java、JavaScript、TypeScript、C#、Go、Rust、PHP、Ruby、Kotlin、C、C++、Shell 指令碼、SQL 和 Scala。雖然本指南著重於 Python 和 Java，例如，這些概念適用於任何支援的程式設計語言。

如何向 Amazon Q Developer 提供內容，以獲得更好的程式碼產生 ？

從現有的程式碼開始，匯入相關的程式庫，建立類別和函數，或建立程式碼骨架。針對自然語言提示使用標準註解區塊。讓您的指令碼專注於特定目標，並將不同的功能模組化為具有相關內容的個別指令碼。如需詳細資訊，請參閱 [Amazon Q Developer 的最佳實務](#)。

如果使用 Amazon Q Developer 產生內嵌程式碼不準確，該怎麼辦 ？

檢閱指令碼的內容，確保程式庫存在，並確保類別和函數與新程式碼相關。將程式碼模組化，並依目標分隔不同的類別和函數。撰寫清楚且特定的提示或註解。如果您仍然不確定程式碼的準確性，而且無法

繼續進行，請透過 Amazon Q 開始聊天，並傳送程式碼片段與指示。如需詳細資訊，請參閱 [Amazon Q Developer 中的程式碼產生案例疑難排解](#)。

如何使用 Amazon Q Developer 聊天功能進程式碼產生和疑難排解？

與 Amazon Q 聊天以產生常用函數、尋求建議或解釋程式碼。如果初始回應不滿意，請實驗不同的提示，並遵循提供的 URLs。此外，向 Amazon Q 提供意見回饋，以協助改善其未來的聊天效能。使用拇指向上和拇指向下圖示來提供意見回饋。如需詳細資訊，請參閱 [聊天範例](#)。

使用 Amazon Q Developer 有哪些最佳實務？

提供相關內容、實驗和重複提示、在接受程式碼前檢閱程式碼建議、使用自訂功能，以及了解資料隱私權和內容使用政策。如需詳細資訊，請參閱 [使用 Amazon Q Developer 產生程式碼的最佳實務](#)，以及 [使用 Amazon Q Developer 產生程式碼建議的最佳實務](#)。

我可以自訂 Amazon Q Developer 來根據自己的程式碼產生建議嗎？

是，使用自訂，這是 Amazon Q Developer 的進階功能。透過自訂，企業可以提供自己的程式碼儲存庫，讓 Amazon Q Developer 能夠建議內嵌程式碼建議。如需詳細資訊，請參閱 [Amazon Q Developer and Resources 的進階功能](#)。 ???

使用 Amazon Q Developer 的後續步驟

透過此完整指南獲得的知識，您可以在編碼工作流程中有效地使用 Amazon Q Developer。在偏好的 IDE ([Visual Studio Code](#) 或 [JetBrains](#)) AWS 工具組 中安裝，並開始探索生成式 AI 支援的程式碼產生和來自 Amazon Q Developer 的建議。

釋放 Amazon Q Developer 完整潛力的最有效方法是使用您自己的程式碼進行實作體驗。當您將 Amazon Q 整合到您的開發生命週期時，請參閱本指南以取得最佳實務、疑難排解和實際範例。

此外，請檢閱 [資源](#) 中參考的 AWS 部落格和開發人員指南，隨時掌握最新資訊。這些資源提供最新的更新、最佳實務和洞見，協助您最佳化 Amazon Q Developer 的使用。

您的意見回饋對於改善本指南並協助它仍是開發人員的寶貴資源非常寶貴。分享未來版本的體驗、挑戰和建議。您的輸入將有助於透過其他範例、故障診斷案例和根據您的需求量身打造的洞察來增強指南。

資源

AWS 部落格

- [使用 Amazon Q 加速您的軟體開發生命週期](#)
- [使用 Amazon Q 開發人員代理程式重新構想軟體開發](#)
- [使用 Amazon Q 的五個疑難排解範例](#)
- [IDE使用私有程式碼基礎在 中自訂 Amazon Q 開發人員](#)
- [適用於程式碼轉換的 Amazon Q 開發人員代理程式加速 Java 升級的三種方式](#)
- [利用 Amazon Q 開發人員進行高效程式碼偵錯和維護](#)
- [使用 Amazon Q Developer 測試您的應用程式](#)

AWS 文件

- [Amazon Q 開發人員使用者指南](#)
- [Amazon Q 開發人員程式碼自訂](#)
- [Amazon Q 開發人員程式碼轉換](#)

AWS 研討會

- [Amazon Q 開發人員沉浸日](#)
- [Amazon Q 開發人員研討會 - 建置 Q-Words 應用程式](#)
- [Amazon Q 開發人員研討會 - 建立有效的提示](#)

貢獻者

下列個人對本指南有所貢獻：

- Joe King，資深資料科學家，AWS
- Prateek Gupta，資深 CAA 團隊負責人 AWS
- Manohar Reddy Arranagu，DevOps 架構師，AWS
- Soumik Roy，雲端應用程式架構師 AWS
- Sanket Shinde，顧問 AWS

文件歷史記錄

下表描述了本指南的重大變更。如果您想要收到未來更新的通知，您可以訂閱[RSS摘要](#)。

變更	描述	日期
初次出版	—	2024 年 8 月 16 日

AWS 規範性指引詞彙表

以下是 AWS Prescriptive Guidance 提供的策略、指南和模式中常用的術語。若要建議項目，請使用詞彙表末尾的提供意見回饋連結。

數字

7 R

將應用程式移至雲端的七種常見遷移策略。這些策略以 Gartner 在 2011 年確定的 5 R 為基礎，包括以下內容：

- 重構/重新架構 – 充分利用雲端原生功能來移動應用程式並修改其架構，以提高敏捷性、效能和可擴展性。這通常涉及移植作業系統和資料庫。範例：將您的現場部署 Oracle 資料庫 遷移至 Amazon Aurora PostgreSQL 相容版本。
- 平台轉換 (隨即重塑) – 將應用程式移至雲端，並引入一定程度的優化以利用雲端功能。範例：將內部部署 Oracle 資料庫 遷移至 中的 Amazon Relational Database Service (Amazon RDS) for Oracle AWS 雲端。
- 重新購買 (捨棄再購買) – 切換至不同的產品，通常從傳統授權移至 SaaS 模型。範例：將您的客戶關係管理 (CRM) 系統 遷移至 Salesforce.com。
- 主機轉換 (隨即轉移) – 將應用程式移至雲端，而不進行任何變更以利用雲端功能。範例：將您的現場部署 Oracle 資料庫 遷移至 中 EC2 執行個體上的 Oracle AWS 雲端。
- 重新放置 (虛擬機器監視器等級隨即轉移) – 將基礎設施移至雲端，無需購買新硬體、重寫應用程式或修改現有操作。您可以將伺服器從內部部署平台遷移到相同平台的雲端服務。範例：將 Microsoft Hyper-V 應用程式 遷移至 AWS。
- 保留 (重新檢視) – 將應用程式保留在來源環境中。其中可能包括需要重要重構的應用程式，且您希望將該工作延遲到以後，以及您想要保留的舊版應用程式，因為沒有業務理由來進行遷移。
- 淘汰 – 解除委任或移除來源環境中不再需要的應用程式。

A

ABAC

請參閱 [屬性型存取控制](#)。

抽象服務

請參閱 [受管服務](#)。

ACID

請參閱 [原子性、一致性、隔離性、持久性](#)。

主動-主動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步 (透過使用雙向複寫工具或雙重寫入操作)，且兩個資料庫都在遷移期間處理來自連接應用程式的交易。此方法支援小型、受控制批次的遷移，而不需要一次性切換。它更靈活，但比 [主動-被動遷移](#) 需要更多的工作。

主動-被動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步，但只有來源資料庫會在資料複寫至目標資料庫時處理來自連線應用程式的交易。目標資料庫在遷移期間不接受任何交易。

彙總函數

在一組資料列上運作的 SQL 函數，會計算群組的單一傳回值。彙總函數的範例包括 SUM 和 MAX。

AI

請參閱 [人工智慧](#)。

AIOps

請參閱 [人工智慧操作](#)。

匿名化

永久刪除資料集中個人資訊的程序。匿名化有助於保護個人隱私權。匿名資料不再被視為個人資料。

反模式

經常性問題的常用解決方案，其中解決方案具有反效益、無效或比替代解決方案效率更低。

應用程式控制

一種安全方法，僅允許使用核准的應用程式，以協助保護系統免受惡意軟體攻擊。

應用程式組合

有關組織使用的每個應用程式的詳細資訊的集合，包括建置和維護應用程式的成本及其商業價值。此資訊是 [產品組合探索和分析程序](#) 的關鍵，有助於識別要遷移、現代化和優化的應用程式並排定其優先順序。

人工智慧 (AI)

電腦科學領域，致力於使用運算技術來執行通常與人類相關的認知功能，例如學習、解決問題和識別模式。如需詳細資訊，請參閱[什麼是人工智慧？](#)

人工智慧操作 (AIOps)

使用機器學習技術解決操作問題、減少操作事件和人工干預以及提高服務品質的程序。如需有關如何在 AWS 遷移策略中使用 AIOps 的詳細資訊，請參閱[操作整合指南](#)。

非對稱加密

一種加密演算法，它使用一對金鑰：一個用於加密的公有金鑰和一個用於解密的私有金鑰。您可以共用公有金鑰，因為它不用於解密，但對私有金鑰存取應受到高度限制。

原子性、一致性、隔離性、持久性 (ACID)

一組軟體屬性，即使在出現錯誤、電源故障或其他問題的情況下，也能確保資料庫的資料有效性和操作可靠性。

屬性型存取控制 (ABAC)

根據使用者屬性 (例如部門、工作職責和團隊名稱) 建立精細許可的實務。如需詳細資訊，請參閱《AWS Identity and Access Management (IAM) 文件》中的[ABAC for AWS](#)。

授權資料來源

存放主要版本資料的位置，被視為最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以處理或修改資料，例如匿名、修訂或假名化資料。

可用區域

中的不同位置 AWS 區域，可隔離其他可用區域中的故障，並提供相同區域中其他可用區域的低成本、低延遲網路連線能力。

AWS 雲端採用架構 (AWS CAF)

的指導方針和最佳實務架構 AWS，可協助組織制定有效率且有效的計劃，以成功移至雲端。AWS CAF 會將指導方針整理成六個重點領域：業務、人員、控管、平台、安全和營運。業務、人員和控管層面著重於業務技能和程序；平台、安全和操作層面著重於技術技能和程序。例如，人員層面針對處理人力資源 (HR)、人員配備功能和人員管理的利害關係人。為此，AWS CAF 為人員開發、訓練和通訊提供指引，協助組織做好成功採用雲端的準備。如需詳細資訊，請參閱[AWS CAF 網站](#)和[AWS CAF 白皮書](#)。

AWS 工作負載資格架構 (AWS WQF)

評估資料庫遷移工作負載、建議遷移策略並提供工作預估值的工具。AWS WQF 隨附於 AWS Schema Conversion Tool (AWS SCT)。它會分析資料庫結構描述和程式碼物件、應用程式程式碼、相依性和效能特性，並提供評估報告。

B

錯誤的機器人

旨在中斷或傷害個人或組織的[機器人](#)。

BCP

請參閱[業務持續性規劃](#)。

行為圖

資源行為的統一互動式檢視，以及一段時間後的互動。您可以將行為圖與 Amazon Detective 搭配使用來檢查失敗的登入嘗試、可疑的 API 呼叫和類似動作。如需詳細資訊，請參閱偵測文件中的[行為圖中的資料](#)。

大端序系統

首先儲存最高有效位元組的系統。另請參閱[Endianness](#)。

二進制分類

預測二進制結果的過程 (兩個可能的類別之一)。例如，ML 模型可能需要預測諸如「此電子郵件是否是垃圾郵件？」等問題或「產品是書還是汽車？」

Bloom 篩選條件

一種機率性、記憶體高效的資料結構，用於測試元素是否為集的成員。

藍/綠部署

一種部署策略，您可以在其中建立兩個不同但相同的環境。您可以在一個環境（藍色）中執行目前的應用程式版本，並在另一個環境（綠色）中執行新的應用程式版本。此策略可協助您快速復原，並將影響降至最低。

機器人

透過網際網路執行自動化任務並模擬人類活動或互動的軟體應用程式。有些機器人有用或有益，例如在網際網路上編製資訊索引的 Web 爬蟲程式。某些其他機器人稱為惡意機器人，旨在中斷或傷害個人或組織。

殭屍網路

受到[惡意軟體](#)感染且受單一方控制之[機器人的](#)網路，稱為機器人繼承器或機器人運算子。殭屍網路是擴展機器人及其影響的最佳已知機制。

分支

程式碼儲存庫包含的區域。儲存庫中建立的第一個分支是主要分支。您可以從現有分支建立新分支，然後在新分支中開發功能或修正錯誤。您建立用來建立功能的分支通常稱為功能分支。當準備好發佈功能時，可以將功能分支合併回主要分支。如需詳細資訊，請參閱[關於分支](#) (GitHub 文件)。

碎片存取

在特殊情況下，以及透過核准的程序，讓使用者快速取得他們通常無權存取 AWS 帳戶 之 的存取權。如需詳細資訊，請參閱 Well-Architected 指南中的 AWS [實作打破玻璃程序](#) 指標。

棕地策略

環境中的現有基礎設施。對系統架構採用棕地策略時，可以根據目前系統和基礎設施的限制來設計架構。如果正在擴展現有基礎設施，則可能會混合棕地和[綠地](#)策略。

緩衝快取

儲存最常存取資料的記憶體區域。

業務能力

業務如何創造價值 (例如，銷售、客戶服務或營銷)。業務能力可驅動微服務架構和開發決策。如需詳細資訊，請參閱在 [AWS 上執行容器化微服務](#) 白皮書的 [圍繞業務能力進行組織](#) 部分。

業務連續性規劃 (BCP)

一種解決破壞性事件 (如大規模遷移) 對營運的潛在影響並使業務能夠快速恢復營運的計畫。

C

CAF

請參閱[AWS 雲端採用架構](#)。

Canary 部署

版本對最終使用者的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前版本。

CCoE

請參閱 [Cloud Center of Excellence](#)。

CDC

請參閱[變更資料擷取](#)。

變更資料擷取 (CDC)

追蹤對資料來源 (例如資料庫表格) 的變更並記錄有關變更改的中繼資料的程序。您可以將 CDC 用於各種用途，例如稽核或複寫目標系統中的變更以保持同步。

混沌工程

故意引入故障或破壞性事件，以測試系統的彈性。您可以使用 [AWS Fault Injection Service \(AWS FIS\)](#) 執行實驗，為您的 AWS 工作負載帶來壓力，並評估其回應。

CI/CD

請參閱[持續整合和持續交付](#)。

分類

有助於產生預測的分類程序。用於分類問題的 ML 模型可預測離散值。離散值永遠彼此不同。例如，模型可能需要評估影像中是否有汽車。

用戶端加密

在目標 AWS 服務接收資料之前，在本機加密資料。

雲端卓越中心 (CCoE)

一個多學科團隊，可推動整個組織的雲端採用工作，包括開發雲端最佳實務、調動資源、制定遷移時間表以及領導組織進行大規模轉型。如需詳細資訊，請參閱 AWS 雲端企業策略部落格上的 [CCoE 文章](#)。

雲端運算

通常用於遠端資料儲存和 IoT 裝置管理的雲端技術。雲端運算通常連接到[邊緣運算](#)技術。

雲端操作模型

在 IT 組織中，用於建置、成熟和最佳化一或多個雲端環境的操作模型。如需詳細資訊，請參閱[建置您的雲端操作模型](#)。

採用雲端階段

組織在遷移至時通常會經歷的四個階段 AWS 雲端：

- 專案 – 執行一些與雲端相關的專案以進行概念驗證和學習用途
- 基礎 – 進行基礎投資以擴展雲端採用 (例如，建立登陸區域、定義 CCoE、建立營運模型)

- 遷移 – 遷移個別應用程式
- 重塑 – 優化產品和服務，並在雲端中創新

這些階段是由 Stephen Orban 在部落格文章 [The Journey Toward Cloud-First 和 Enterprise Strategy 部落格上的採用階段](#) 中所定義。AWS 雲端 如需有關它們如何與 AWS 遷移策略關聯的資訊，請參閱 [遷移整備指南](#)。

CMDB

請參閱 [組態管理資料庫](#)。

程式碼儲存庫

透過版本控制程序來儲存及更新原始程式碼和其他資產 (例如文件、範例和指令碼) 的位置。常見的雲端儲存庫包括 GitHub 或 Bitbucket Cloud。程式碼的每個版本都稱為分支。在微服務結構中，每個儲存庫都專用於單個功能。單一 CI/CD 管道可以使用多個儲存庫。

冷快取

一種緩衝快取，它是空的、未填充的，或者包含過時或不相關的資料。這會影響效能，因為資料庫執行個體必須從主記憶體或磁碟讀取，這比從緩衝快取讀取更慢。

冷資料

很少存取且通常是歷史資料的資料。查詢這類資料時，通常可接受慢查詢。將此資料移至效能較低且成本較低的儲存層或類別，可以降低成本。

電腦視覺 (CV)

使用機器學習從數位影像和影片等視覺化格式分析和擷取資訊的 [AI](#) 欄位。例如，Amazon SageMaker AI 提供 CV 的影像處理演算法。

組態偏離

對於工作負載，組態會從預期狀態變更。這可能會導致工作負載不合規，而且通常是漸進和無意的。

組態管理資料庫 (CMDB)

儲存和管理有關資料庫及其 IT 環境的資訊的儲存庫，同時包括硬體和軟體元件及其組態。您通常在遷移的產品組合探索和分析階段使用 CMDB 中的資料。

一致性套件

您可以組合的 AWS Config 規則和修補動作集合，以自訂您的合規和安全檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和 區域中或整個組織的單一實體。如需詳細資訊，請參閱 AWS Config 文件中的 [一致性套件](#)。

持續整合和持續交付 (CI/CD)

自動化軟體發程序的來源、建置、測試、暫存和生產階段的程序。CI/CD 通常被描述為管道。CI/CD 可協助您將程序自動化、提升生產力、改善程式碼品質以及加快交付速度。如需詳細資訊，請參閱[持續交付的優點](#)。CD 也可表示持續部署。如需詳細資訊，請參閱[持續交付與持續部署](#)。

CV

請參閱[電腦視覺](#)。

D

靜態資料

網路中靜止的資料，例如儲存中的資料。

資料分類

根據重要性和敏感性來識別和分類網路資料的程序。它是所有網路安全風險管理策略的關鍵組成部分，因為它可以協助您確定適當的資料保護和保留控制。資料分類是 AWS Well-Architected Framework 中安全支柱的元件。如需詳細資訊，請參閱[資料分類](#)。

資料偏離

生產資料與用於訓練 ML 模型的資料之間有意義的變化，或輸入資料隨時間有意義的變更。資料偏離可以降低 ML 模型預測的整體品質、準確性和公平性。

傳輸中的資料

在您的網路中主動移動的資料，例如在網路資源之間移動。

資料網格

架構架構，提供分散式、分散式資料擁有權與集中式管理。

資料最小化

僅收集和處理嚴格必要資料的原則。在中實作資料最小化 AWS 雲端可以降低隱私權風險、成本和分析碳足跡。

資料周邊

AWS 環境中的一組預防性防護機制，可協助確保只有信任的身分才能從預期的網路存取信任的資源。如需詳細資訊，請參閱[在上建置資料周邊 AWS](#)。

資料預先處理

將原始資料轉換成 ML 模型可輕鬆剖析的格式。預處理資料可能意味著移除某些欄或列，並解決遺失、不一致或重複的值。

資料來源

在整個生命週期中追蹤資料的原始伺服器 and 歷史記錄的程序，例如資料的產生、傳輸和儲存方式。

資料主體

正在收集和處理資料的個人。

資料倉儲

支援商業智慧的資料管理系統，例如分析。資料倉儲通常包含大量歷史資料，通常用於查詢和分析。

資料庫定義語言 (DDL)

用於建立或修改資料庫中資料表和物件之結構的陳述式或命令。

資料庫處理語言 (DML)

用於修改 (插入、更新和刪除) 資料庫中資訊的陳述式或命令。

DDL

請參閱[資料庫定義語言](#)。

深度整體

結合多個深度學習模型進行預測。可以使用深度整體來獲得更準確的預測或估計預測中的不確定性。

深度學習

一個機器學習子領域，它使用多層人工神經網路來識別感興趣的輸入資料與目標變數之間的對應關係。

深度防禦

這是一種資訊安全方法，其中一系列的安全機制和控制項會在整個電腦網路中精心分層，以保護網路和其中資料的機密性、完整性和可用性。當您在上採用此策略時 AWS，您可以在 AWS Organizations 結構的不同層新增多個控制項，以協助保護資源。例如，defense-in-depth 方法可能會結合多重驗證、網路分割和加密。

委派的管理員

在中 AWS Organizations，相容的服務可以註冊 AWS 成員帳戶，以管理組織的帳戶和管理該服務的許可。此帳戶稱為該服務的委派管理員。如需詳細資訊和相容服務清單，請參閱 AWS Organizations 文件中的[可搭配 AWS Organizations運作的服務](#)。

deployment

在目標環境中提供應用程式、新功能或程式碼修正的程序。部署涉及在程式碼庫中實作變更，然後在應用程式環境中建置和執行該程式碼庫。

開發環境

請參閱[環境](#)。

偵測性控制

一種安全控制，用於在事件發生後偵測、記錄和提醒。這些控制是第二道防線，提醒您注意繞過現有預防性控制的安全事件。如需詳細資訊，請參閱在 AWS 上實作安全控制中的[偵測性控制](#)。

開發值串流映射 (DVSM)

一種程序，用於識別並優先考慮對軟體開發生命週期中的速度和品質造成負面影響的限制。DVSM 延伸了原本專為精簡製造實務設計的價值串流映射程序。它著重於透過軟體開發程序建立和移動價值所需的步驟和團隊。

數位分身

真實世界系統的虛擬呈現，例如建築物、工廠、工業設備或生產線。數位分身支援預測性維護、遠端監控和生產最佳化。

維度資料表

在[星星結構描述](#)中，較小的資料表包含有關事實資料表中量化資料的資料屬性。維度資料表屬性通常是文字欄位或離散數字，其行為與文字相似。這些屬性通常用於查詢限制、篩選和結果集標記。

災難

防止工作負載或系統在其主要部署位置實現其業務目標的事件。這些事件可能是自然災難、技術故障或人為動作的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您用來將[災難](#)造成的停機時間和資料遺失降至最低的策略和程序。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的[上工作負載的災難復原 AWS：雲端中的復原](#)。

DML

請參閱[資料庫處理語言](#)。

領域驅動的設計

一種開發複雜軟體系統的方法，它會將其元件與每個元件所服務的不斷發展的領域或核心業務目標相關聯。Eric Evans 在其著作 *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003) 中介紹了這一概念。如需有關如何將領域驅動的設計與 strangler fig 模式搭配使用的資訊，請參閱[使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET \(ASMX\) Web 服務](#)。

DR

請參閱[災難復原](#)。

偏離偵測

追蹤與基準組態的偏差。例如，您可以使用 AWS CloudFormation 來偵測系統資源中的偏離，也可以使用 AWS Control Tower 來[偵測登陸區域中可能影響控管要求合規性的變更](#)。<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html>

DVSM

請參閱[開發值串流映射](#)。

E

EDA

請參閱[探索性資料分析](#)。

EDI

請參閱[電子資料交換](#)。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與[雲端運算](#)相比，邊緣運算可以減少通訊延遲並改善回應時間。

電子資料交換 (EDI)

在組織之間自動交換商業文件。如需詳細資訊，請參閱[什麼是電子資料交換](#)。

加密

將人類可讀取的純文字資料轉換為加密文字的運算程序。

加密金鑰

由加密演算法產生的隨機位元的加密字串。金鑰長度可能有所不同，每個金鑰的設計都是不可預測且唯一的。

端序

位元組在電腦記憶體中的儲存順序。大端序系統首先儲存最高有效位元組。小端序系統首先儲存最低有效位元組。

端點

請參閱 [服務端點](#)。

端點服務

您可以在虛擬私有雲端 (VPC) 中託管以與其他使用者共用的服務。您可以使用 [建立端點服務](#)，AWS PrivateLink 並將許可授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 委託人。這些帳戶或主體可以透過建立介面 VPC 端點私下連接至您的端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud (Amazon VPC) 文件中的 [建立端點服務](#)。

企業資源規劃 (ERP)

一種系統，可自動化和管理企業的關鍵業務流程（例如會計、[MES](#) 和專案管理）。

信封加密

使用另一個加密金鑰對某個加密金鑰進行加密的程序。如需詳細資訊，請參閱 AWS Key Management Service (AWS KMS) 文件中的 [信封加密](#)。

環境

執行中應用程式的執行個體。以下是雲端運算中常見的環境類型：

- 開發環境 – 執行中應用程式的執行個體，只有負責維護應用程式的核心團隊才能使用。開發環境用來測試變更，然後再將開發環境提升到較高的環境。此類型的環境有時稱為測試環境。
- 較低的環境 – 應用程式的所有開發環境，例如用於初始建置和測試的開發環境。
- 生產環境 – 最終使用者可以存取的執行中應用程式的執行個體。在 CI/CD 管道中，生產環境是最後一個部署環境。
- 較高的環境 – 核心開發團隊以外的使用者可存取的所有環境。這可能包括生產環境、生產前環境以及用於使用者接受度測試的環境。

epic

在敏捷方法中，有助於組織工作並排定工作優先順序的功能類別。epic 提供要求和實作任務的高層級描述。例如，AWS CAF 安全概念包括身分和存取管理、偵測控制、基礎設施安全、資料保護和事件回應。如需有關 AWS 遷移策略中的 Epic 的詳細資訊，請參閱[計畫實作指南](#)。

ERP

請參閱[企業資源規劃](#)。

探索性資料分析 (EDA)

分析資料集以了解其主要特性的過程。您收集或彙總資料，然後執行初步調查以尋找模式、偵測異常並檢查假設。透過計算摘要統計並建立資料可視化來執行 EDA。

F

事實資料表

[星狀結構描述](#)中的中央資料表。它存放有關業務操作的量化資料。一般而言，事實資料表包含兩種類型的資料欄：包含度量的資料，以及包含維度資料表外部索引鍵的資料欄。

快速失敗

一種使用頻繁和增量測試來縮短開發生命週期的理念。這是敏捷方法的關鍵部分。

故障隔離界限

在中 AWS 雲端，像是可用區域 AWS 區域、控制平面或資料平面等邊界會限制故障的影響，並有助於改善工作負載的彈性。如需詳細資訊，請參閱[AWS 故障隔離界限](#)。

功能分支

請參閱[分支](#)。

特徵

用來進行預測的輸入資料。例如，在製造環境中，特徵可能是定期從製造生產線擷取的影像。

功能重要性

特徵對於模型的預測有多重要。這通常表示為可以透過各種技術來計算的數值得分，例如 Shapley Additive Explanations (SHAP) 和積分梯度。如需詳細資訊，請參閱[機器學習模型可解譯性 AWS](#)。

特徵轉換

優化 ML 程序的資料，包括使用其他來源豐富資料、調整值、或從單一資料欄位擷取多組資訊。這可讓 ML 模型從資料中受益。例如，如果將「2021-05-27 00:15:37」日期劃分為「2021」、「五月」、「週四」和「15」，則可以協助學習演算法學習與不同資料元件相關聯的細微模式。

少量擷取提示

在要求 [LLM](#) 執行類似的任務之前，提供少量示範任務和所需輸出的範例。此技術是內容內學習的應用程式，其中模型會從內嵌在提示中的範例 (快照) 中學習。對於需要特定格式、推理或網域知識的任務，少量的提示非常有效。另請參閱[零鏡頭提示](#)。

FGAC

請參閱[精細存取控制](#)。

精細存取控制 (FGAC)

使用多個條件來允許或拒絕存取請求。

閃切遷移

一種資料庫遷移方法，透過[變更資料擷取](#)使用連續資料複寫，以盡可能在最短的時間內遷移資料，而不是使用分階段方法。目標是將停機時間降至最低。

FM

請參閱[基礎模型](#)。

基礎模型 (FM)

大型深度學習神經網路，已針對廣義和未標記資料的大量資料集進行訓練。FMs 能夠執行各種一般任務，例如了解語言、產生文字和影像，以及以自然語言交談。如需詳細資訊，請參閱[什麼是基礎模型](#)。

G

生成式 AI

已針對大量資料進行訓練的 [AI](#) 模型子集，可使用簡單的文字提示建立新的內容和成品，例如影像、影片、文字和音訊。如需詳細資訊，請參閱[什麼是生成式 AI](#)。

地理封鎖

請參閱[地理限制](#)。

地理限制 (地理封鎖)

Amazon CloudFront 中的選項，可防止特定國家/地區的使用者存取內容分發。您可以使用允許清單或封鎖清單來指定核准和禁止的國家/地區。如需詳細資訊，請參閱 CloudFront 文件中的[限制內容的地理分佈](#)。

Gitflow 工作流程

這是一種方法，其中較低和較高環境在原始碼儲存庫中使用不同分支。Gitflow 工作流程被視為舊版，而以[幹線為基礎的工作流程](#)是現代、偏好的方法。

黃金影像

系統或軟體的快照，做為部署該系統或軟體新執行個體的範本。例如，在製造中，黃金映像可用於在多個裝置上佈建軟體，並有助於提高裝置製造操作的速度、可擴展性和生產力。

綠地策略

新環境中缺乏現有基礎設施。對系統架構採用綠地策略時，可以選擇所有新技術，而不會限制與現有基礎設施的相容性，也稱為[棕地](#)。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

防護機制

有助於跨組織單位 (OU) 來管控資源、政策和合規的高層級規則。預防性防護機制會強制執行政策，以確保符合合規標準。透過使用服務控制政策和 IAM 許可界限來將其實施。偵測性防護機制可偵測政策違規和合規問題，並產生提醒以便修正。它們是透過使用 AWS Config、AWS Security Hub、CSPM、Amazon GuardDuty、Amazon Inspector、AWS Trusted Advisor 和自訂 AWS Lambda 檢查來實施。

H

HA

請參閱[高可用性](#)。

異質資料庫遷移

將來源資料庫遷移至使用不同資料庫引擎的目標資料庫 (例如，Oracle 至 Amazon Aurora)。異質遷移通常是重新架構工作的一部分，而轉換結構描述可能是一項複雜任務。[AWS 提供有助於結構描述轉換的 AWS SCT](#)。

高可用性 (HA)

在遇到挑戰或災難時，工作負載能夠在不介入的情況下持續運作。HA 系統的設計目的是自動容錯移轉、持續提供高品質的效能，以及處理不同的負載和故障，並將效能影響降至最低。

歷史現代化

一種方法，用於現代化和升級操作技術 (OT) 系統，以更好地滿足製造業的需求。歷史資料是一種資料庫，用於從工廠中的各種來源收集和存放資料。

保留資料

從用於訓練機器學習模型的資料集中保留的部分歷史標記資料。您可以使用保留資料，透過比較模型預測與保留資料來評估模型效能。

異質資料庫遷移

將您的來源資料庫遷移至共用相同資料庫引擎的目標資料庫 (例如，Microsoft SQL Server 至 Amazon RDS for SQL Server)。同質遷移通常是主機轉換或平台轉換工作的一部分。您可以使用原生資料庫公用程式來遷移結構描述。

熱資料

經常存取的資料，例如即時資料或最近的轉譯資料。此資料通常需要高效能儲存層或類別，才能提供快速的查詢回應。

修補程序

緊急修正生產環境中的關鍵問題。由於其緊迫性，通常會在典型 DevOps 發行工作流程之外執行修補程式。

超級護理期間

在切換後，遷移團隊在雲端管理和監控遷移的應用程式以解決任何問題的時段。通常，此期間的長度為 1-4 天。在超級護理期間結束時，遷移團隊通常會將應用程式的責任轉移給雲端營運團隊。

I

IaC

將[基礎設施視為程式碼](#)。

身分型政策

連接至一或多個 IAM 主體的政策，可定義其在 AWS 雲端環境中的許可。

閒置應用程式

90 天期間 CPU 和記憶體平均使用率在 5% 至 20% 之間的應用程式。在遷移專案中，通常會淘汰這些應用程式或將其保留在內部部署。

IloT

請參閱[工業物聯網](#)。

不可變的基礎設施

為生產工作負載部署新基礎設施的模型，而不是更新、修補或修改現有的基礎設施。不可變基礎設施本質上比[可變基礎設施](#)更一致、可靠且可預測。如需詳細資訊，請參閱 AWS Well-Architected Framework [中的使用不可變基礎設施的部署](#)最佳實務。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，接受、檢查和路由來自應用程式外部之網路連線的 VPC。[AWS 安全參考架構](#)建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

增量遷移

一種切換策略，您可以在其中將應用程式分成小部分遷移，而不是執行單一、完整的切換。例如，您最初可能只將一些微服務或使用者移至新系統。確認所有項目都正常運作之後，您可以逐步移動其他微服務或使用者，直到可以解除委任舊式系統。此策略可降低與大型遷移關聯的風險。

工業 4.0

由 [Klaus Schwab](#) 於 2016 年推出的術語，透過連線能力、即時資料、自動化、分析和 AI/ML 的進展，指製造程序的現代化。

基礎設施

應用程式環境中包含的所有資源和資產。

基礎設施即程式碼 (IaC)

透過一組組態檔案來佈建和管理應用程式基礎設施的程序。IaC 旨在協助您集中管理基礎設施，標準化資源並快速擴展，以便新環境可重複、可靠且一致。

工業物聯網 (IIoT)

在製造業、能源、汽車、醫療保健、生命科學和農業等產業領域使用網際網路連線的感測器和裝置。如需詳細資訊，請參閱[建立工業物聯網 \(IIoT\) 數位轉型策略](#)。

檢查 VPC

在 AWS 多帳戶架構中，集中式 VPC 可管理 VPCs 之間（在相同或不同的 AWS 區域）、網際網路和內部部署網路之間的網路流量檢查。[AWS 安全參考架構](#)建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

物聯網 (IoT)

具有內嵌式感測器或處理器的相連實體物體網路，其透過網際網路或本地通訊網路與其他裝置和系統進行通訊。如需詳細資訊，請參閱[什麼是 IoT？](#)

可解釋性

機器學習模型的一個特徵，描述了人類能夠理解模型的預測如何依賴於其輸入的程度。如需詳細資訊，請參閱[的機器學習模型可解釋性 AWS](#)。

IoT

請參閱[物聯網](#)。

IT 資訊庫 (ITIL)

一組用於交付 IT 服務並使這些服務與業務需求保持一致的最佳實務。ITIL 為 ITSM 提供了基礎。

IT 服務管理 (ITSM)

與組織的設計、實作、管理和支援 IT 服務關聯的活動。如需有關將雲端操作與 ITSM 工具整合的資訊，請參閱[操作整合指南](#)。

ITIL

請參閱[IT 資訊庫](#)。

ITSM

請參閱[IT 服務管理](#)。

L

標籤型存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中使用者和資料本身都會獲得明確指派的安全標籤值。使用者安全標籤和資料安全標籤之間的交集會決定使用者可以看到哪些資料列和資料欄。

登陸區域

登陸區域是架構良好的多帳戶 AWS 環境，可擴展且安全。這是一個起點，您的組織可以從此起點快速啟動和部署工作負載與應用程式，並對其安全和基礎設施環境充滿信心。如需有關登陸區域的詳細資訊，請參閱[設定安全且可擴展的多帳戶 AWS 環境](#)。

大型語言模型 (LLM)

預先訓練大量資料的深度學習 [AI](#) 模型。LLM 可以執行多個任務，例如回答問題、摘要文件、將文字翻譯成其他語言，以及完成句子。如需詳細資訊，請參閱[什麼是 LLMs](#)。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱[標籤型存取控制](#)。

最低權限

授予執行任務所需之最低許可的安全最佳實務。如需詳細資訊，請參閱 IAM 文件中的[套用最低權限許可](#)。

隨即轉移

請參閱 [7 個 R](#)。

小端序系統

首先儲存最低有效位元組的系統。另請參閱 [Endianness](#)。

LLM

請參閱[大型語言模型](#)。

較低的環境

請參閱 [環境](#)。

M

機器學習 (ML)

一種使用演算法和技術進行模式識別和學習的人工智慧。機器學習會進行分析並從記錄的資料 (例如物聯網 (IoT) 資料) 中學習，以根據模式產生統計模型。如需詳細資訊，請參閱[機器學習](#)。

主要分支

請參閱[分支](#)。

惡意軟體

旨在危及電腦安全或隱私權的軟體。惡意軟體可能會中斷電腦系統、洩露敏感資訊，或取得未經授權的存取。惡意軟體的範例包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄器。

受管服務

AWS 服務會 AWS 操作基礎設施層、作業系統和平台，而您會存取端點來存放和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也稱為抽象服務。

製造執行系統 (MES)

一種軟體系統，用於追蹤、監控、記錄和控制生產程序，將原物料轉換為現場成品。

MAP

請參閱[遷移加速計劃](#)。

機制

建立工具、推動工具採用，然後檢查結果以進行調整的完整程序。機制是在操作時強化和改善自身的循環。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的[建置機制](#)。

成員帳戶

除了屬於組織一部分的管理帳戶 AWS 帳戶 之外的所有 AWS Organizations。帳戶一次只能是一個組織的成員。

製造執行系統

請參閱[製造執行系統](#)。

訊息佇列遙測傳輸 (MQTT)

根據[發佈/訂閱](#)模式的輕量型machine-to-machine(M2M) 通訊協定，適用於資源受限的 [IoT](#) 裝置。

微服務

一種小型的獨立服務，它可透過定義明確的 API 進行通訊，通常由小型獨立團隊擁有。例如，保險系統可能包含對應至業務能力 (例如銷售或行銷) 或子領域 (例如購買、索賠或分析) 的微服務。微服務的優點包括靈活性、彈性擴展、輕鬆部署、可重複使用的程式碼和適應力。如需詳細資訊，請參閱[使用無 AWS 伺服器服務整合微服務](#)。

微服務架構

一種使用獨立元件來建置應用程式的方法，這些元件會以微服務形式執行每個應用程式程序。這些微服務會使用輕量型 API，透過明確定義的介面進行通訊。此架構中的每個微服務都可以進行更新、部署和擴展，以滿足應用程式特定功能的需求。如需詳細資訊，請參閱[實作微服務 AWS](#)。

Migration Acceleration Program (MAP)

此 AWS 計畫提供諮詢支援、訓練和服務，以協助組織建立強大的營運基礎，以移至雲端，並協助抵銷遷移的初始成本。MAP 包括用於有條不紊地執行舊式遷移的遷移方法以及一組用於自動化和加速常見遷移案例的工具。

大規模遷移

將大部分應用程式組合依波次移至雲端的程序，在每個波次中，都會以更快的速度移動更多應用程式。此階段使用從早期階段學到的最佳實務和經驗教訓來實作團隊、工具和流程的遷移工廠，以透過自動化和敏捷交付簡化工作負載的遷移。這是[AWS 遷移策略](#)的第三階段。

遷移工廠

可透過自動化、敏捷的方法簡化工作負載遷移的跨職能團隊。遷移工廠團隊通常包括營運、業務分析師和擁有者、遷移工程師、開發人員以及從事 Sprint 工作的 DevOps 專業人員。20% 至 50% 之間的企業應用程式組合包含可透過工廠方法優化的重複模式。如需詳細資訊，請參閱此內容集中的[遷移工廠的討論](#)和[雲端遷移工廠指南](#)。

遷移中繼資料

有關完成遷移所需的應用程式和伺服器的資訊。每種遷移模式都需要一組不同的遷移中繼資料。遷移中繼資料的範例包括目標子網路、安全群組和 AWS 帳戶。

遷移模式

可重複的遷移任務，詳細描述遷移策略、遷移目的地以及所使用的遷移應用程式或服務。範例：使用 AWS Application Migration Service 重新託管遷移至 Amazon EC2。

遷移組合評定 (MPA)

線上工具，提供驗證商業案例以遷移至的資訊 AWS 雲端。MPA 提供詳細的組合評定 (伺服器適當規模、定價、總體擁有成本比較、遷移成本分析) 以及遷移規劃 (應用程式資料分析和資料收集、應用程式分組、遷移優先順序，以及波次規劃)。[MPA 工具](#) (需要登入) 可供所有 AWS 顧問和 APN 合作夥伴顧問免費使用。

遷移準備程度評定 (MRA)

使用 AWS CAF 取得組織雲端整備狀態的洞見、識別優缺點，以及建立行動計劃以消除已識別差距的程序。如需詳細資訊，請參閱[遷移準備程度指南](#)。MRA 是[AWS 遷移策略](#)的第一階段。

遷移策略

用來將工作負載遷移至的方法 AWS 雲端。如需詳細資訊，請參閱本詞彙表中的 [7 個 Rs](#) 項目，並請參閱[動員您的組織以加速大規模遷移](#)。

機器學習 (ML)

請參閱[機器學習](#)。

現代化

將過時的 (舊版或單一) 應用程式及其基礎架構轉換為雲端中靈活、富有彈性且高度可用的系統，以降低成本、提高效率並充分利用創新。如需詳細資訊，請參閱 [《》中的現代化應用程式的策略 AWS 雲端](#)。

現代化準備程度評定

這項評估可協助判斷組織應用程式的現代化準備程度；識別優點、風險和相依性；並確定組織能夠在多大程度上支援這些應用程式的未來狀態。評定的結果就是目標架構的藍圖、詳細說明現代化程序的開發階段和里程碑的路線圖、以及解決已發現的差距之行動計畫。如需詳細資訊，請參閱 [《》中的評估應用程式的現代化準備 AWS 雲端](#) 程度。

單一應用程式 (單一)

透過緊密結合的程序作為單一服務執行的應用程式。單一應用程式有幾個缺點。如果一個應用程式功能遇到需求激增，則必須擴展整個架構。當程式碼庫增長時，新增或改進單一應用程式的功能也會變得更加複雜。若要解決這些問題，可以使用微服務架構。如需詳細資訊，請參閱[將單一體系分解為微服務](#)。

MPA

請參閱[遷移產品組合評估](#)。

MQTT

請參閱[訊息佇列遙測傳輸](#)。

多類別分類

一個有助於產生多類別預測的過程 (預測兩個以上的結果之一)。例如，機器學習模型可能會詢問「此產品是書籍、汽車還是電話？」或者「這個客戶對哪種產品類別最感興趣？」

可變基礎設施

更新和修改生產工作負載現有基礎設施的模型。為了提高一致性、可靠性和可預測性，AWS Well-Architected Framework 建議使用[不可變基礎設施](#)做為最佳實務。

O

OAC

請參閱[原始存取控制](#)。

OAI

請參閱[原始存取身分](#)。

OCM

請參閱[組織變更管理](#)。

離線遷移

一種遷移方法，可在遷移過程中刪除來源工作負載。此方法涉及延長停機時間，通常用於小型非關鍵工作負載。

OI

請參閱[操作整合](#)。

OLA

請參閱[操作層級協議](#)。

線上遷移

一種遷移方法，無需離線即可將來源工作負載複製到目標系統。連接至工作負載的應用程式可在遷移期間繼續運作。此方法涉及零至最短停機時間，通常用於關鍵的生產工作負載。

OPC-UA

請參閱[開啟程序通訊 - 統一架構](#)。

開放程序通訊 - 統一架構 (OPC-UA)

用於工業自動化machine-to-machine(M2M) 通訊協定。OPC-UA 提供資料加密、身分驗證和授權機制的互通性標準。

操作水準協議 (OLA)

一份協議，闡明 IT 職能群組承諾向彼此提供的內容，以支援服務水準協議 (SLA)。

操作整備審查 (ORR)

問題和相關最佳實務的檢查清單，可協助您了解、評估、預防或減少事件和可能失敗的範圍。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的[操作準備度審查 \(ORR\)](#)。

操作技術 (OT)

使用實體環境控制工業操作、設備和基礎設施的硬體和軟體系統。在製造中，OT 和資訊技術 (IT) 系統的整合是[工業 4.0](#) 轉型的關鍵重點。

操作整合 (OI)

在雲端中將操作現代化的程序，其中包括準備程度規劃、自動化和整合。如需詳細資訊，請參閱[操作整合指南](#)。

組織追蹤

建立的線索 AWS CloudTrail 會記錄 AWS 帳戶 組織中所有 的所有事件 AWS Organizations。在屬於組織的每個 AWS 帳戶 中建立此追蹤，它會跟蹤每個帳戶中的活動。如需詳細資訊，請參閱 CloudTrail 文件中的[建立組織追蹤](#)。

組織變更管理 (OCM)

用於從人員、文化和領導力層面管理重大、顛覆性業務轉型的架構。OCM 透過加速變更採用、解決過渡問題，以及推動文化和組織變更，協助組織為新系統和策略做好準備，並轉移至新系統和策略。在 AWS 遷移策略中，此架構稱為人員加速，因為雲端採用專案所需的變更速度。如需詳細資訊，請參閱[OCM 指南](#)。

原始存取控制 (OAC)

CloudFront 中的增強型選項，用於限制存取以保護 Amazon Simple Storage Service (Amazon S3) 內容。OAC 支援所有 S3 儲存貯體中的所有伺服器端加密 AWS KMS (SSE-KMS) AWS 區域，以及對 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

CloudFront 中的一個選項，用於限制存取以保護 Amazon S3 內容。當您使用 OAI 時，CloudFront 會建立一個可供 Amazon S3 進行驗證的主體。經驗證的主體只能透過特定 CloudFront 分發來存取 S3 儲存貯體中的內容。另請參閱[OAC](#)，它可提供更精細且增強的存取控制。

ORR

請參閱[操作整備審核](#)。

OT

請參閱[操作技術](#)。

傳出 (輸出) VPC

在 AWS 多帳戶架構中，處理從應用程式內啟動之網路連線的 VPC。[AWS 安全參考架構](#)建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

P

許可界限

附接至 IAM 主體的 IAM 管理政策，可設定使用者或角色擁有的最大許可。如需詳細資訊，請參閱 IAM 文件中的[許可界限](#)。

個人身分識別資訊 (PII)

直接檢視或與其他相關資料配對時，可用來合理推斷個人身分的資訊。PII 的範例包括名稱、地址和聯絡資訊。

PII

請參閱[個人身分識別資訊](#)。

手冊

一組預先定義的步驟，可擷取與遷移關聯的工作，例如在雲端中提供核心操作功能。手冊可以採用指令碼、自動化執行手冊或操作現代化環境所需的程序或步驟摘要的形式。

PLC

請參閱[可程式設計邏輯控制器](#)。

PLM

請參閱[產品生命週期管理](#)。

政策

可定義許可的物件（請參閱[身分型政策](#)）、指定存取條件（請參閱[資源型政策](#)），或定義組織中所有帳戶的最大許可 AWS Organizations（請參閱[服務控制政策](#)）。

混合持久性

根據資料存取模式和其他需求，獨立選擇微服務的資料儲存技術。如果您的微服務具有相同的資料儲存技術，則其可能會遇到實作挑戰或效能不佳。如果微服務使用最適合其需求的資料儲存，則可以更輕鬆地實作並達到更好的效能和可擴展性。

組合評定

探索、分析應用程式組合並排定其優先順序以規劃遷移的程序。如需詳細資訊，請參閱[評估遷移準備程度](#)。

述詞

傳回 true 或的查詢條件 false，通常位於 WHERE 子句中。

述詞下推

一種資料庫查詢最佳化技術，可在傳輸前篩選查詢中的資料。這可減少必須從關聯式資料庫擷取和處理的資料量，並改善查詢效能。

預防性控制

旨在防止事件發生的安全控制。這些控制是第一道防線，可協助防止對網路的未經授權存取或不必要變更。如需詳細資訊，請參閱在 AWS 上實作安全控制中的[預防性控制](#)。

委託人

中可執行動作和存取資源 AWS 的實體。此實體通常是 AWS 帳戶、IAM 角色或使用者的根使用者。如需詳細資訊，請參閱 IAM 文件中[角色術語和概念](#)中的主體。

依設計的隱私權

透過整個開發程序將隱私權納入考量的系統工程方法。

私有託管區域

一種容器，它包含有關您希望 Amazon Route 53 如何回應一個或多個 VPC 內的域及其子域之 DNS 查詢的資訊。如需詳細資訊，請參閱 Route 53 文件中的[使用私有託管區域](#)。

主動控制

旨在防止部署不合規資源的[安全控制](#)。這些控制項會在佈建資源之前對其進行掃描。如果資源不符合控制項，則不會佈建。如需詳細資訊，請參閱 AWS Control Tower 文件中的[控制項參考指南](#)，並參閱實作安全[控制項中的主動](#)控制項。 AWS

產品生命週期管理 (PLM)

產品整個生命週期的資料和程序管理，從設計、開發和啟動，到成長和成熟，再到拒絕和移除。

生產環境

請參閱[環境](#)。

可程式設計邏輯控制器 (PLC)

在製造中，高度可靠、可調整的電腦，可監控機器並自動化製造程序。

提示鏈結

使用一個 [LLM](#) 提示的輸出做為下一個提示的輸入，以產生更好的回應。此技術用於將複雜任務分解為子任務，或反覆精簡或展開初步回應。它有助於提高模型回應的準確性和相關性，並允許更精細、個人化的結果。

擬匿名化

將資料集中的個人識別符取代為預留位置值的程序。假名化有助於保護個人隱私權。假名化資料仍被視為個人資料。

發佈/訂閱 (pub/sub)

一種模式，可啟用微服務之間的非同步通訊，以提高可擴展性和回應能力。例如，在微服務型 [MES](#) 中，微服務可以將事件訊息發佈到其他微服務可訂閱的頻道。系統可以新增新的微服務，而無需變更發佈服務。

Q

查詢計劃

一系列步驟，如指示，用於存取 SQL 關聯式資料庫系統中的資料。

查詢計劃迴歸

在資料庫服務優化工具選擇的計畫比對資料庫環境進行指定的變更之前的計畫不太理想時。這可能因為對統計資料、限制條件、環境設定、查詢參數繫結的變更以及資料庫引擎的更新所導致。

R

RACI 矩陣

請參閱 [負責、負責、諮詢、告知 \(RACI\)](#)。

RAG

請參閱 [擷取增強生成](#)。

勒索軟體

一種惡意軟體，旨在阻止對計算機系統或資料的存取，直到付款為止。

RASCI 矩陣

請參閱[負責、負責、諮詢、告知 \(RACI\)](#)。

RCAC

請參閱[資料列和資料欄存取控制](#)。

僅供讀取複本

用於唯讀用途的資料庫複本。您可以將查詢路由至僅供讀取複本以減少主資料庫的負載。

重新架構師

請參閱[7 個 R](#)。

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這會決定最後一個復原點與服務中斷之間可接受的資料遺失。

復原時間目標 (RTO)

服務中斷與服務還原之間的可接受延遲上限。

重構

請參閱[7 個 R](#)。

區域

地理區域中的 AWS 資源集合。每個 AWS 區域 都獨立於其他，以提供容錯能力、穩定性和彈性。如需詳細資訊，請參閱[指定 AWS 區域 您的帳戶可以使用哪些](#)。

迴歸

預測數值的 ML 技術。例如，為了解決「這房子會賣什麼價格？」的問題 ML 模型可以使用線性迴歸模型，根據已知的房屋事實 (例如，平方英尺) 來預測房屋的銷售價格。

重新託管

請參閱[7 個 R](#)。

版本

在部署程序中，它是將變更提升至生產環境的動作。

重新定位

請參閱 [7 個 R](#)。

Replatform

請參閱 [7 個 R](#)。

回購

請參閱 [7 個 R](#)。

彈性

應用程式抵禦中斷或從中斷中復原的能力。[在中規劃彈性時，高可用性和災難復原](#)是常見的考量 AWS 雲端。如需詳細資訊，請參閱[AWS 雲端 彈性](#)。

資源型政策

附接至資源的政策，例如 Amazon S3 儲存貯體、端點或加密金鑰。這種類型的政策會指定允許存取哪些主體、支援的動作以及必須滿足的任何其他條件。

負責者、當責者、事先諮詢者和事後告知者 (RACI) 矩陣

矩陣，定義所有參與遷移活動和雲端操作之各方的角色和責任。矩陣名稱衍生自矩陣中定義的責任類型：負責人 (R)、責任 (A)、已諮詢 (C) 和知情 (I)。支援 (S) 類型為選用。如果您包含支援，則矩陣稱為 RASCI 矩陣，如果您排除它，則稱為 RACI 矩陣。

回應性控制

一種安全控制，旨在驅動不良事件或偏離安全基準的補救措施。如需詳細資訊，請參閱在 AWS 上實作安全控制中的[回應性控制](#)。

保留

請參閱 [7 個 R](#)。

淘汰

請參閱 [7 個 R](#)。

檢索增強生成 (RAG)

[一種生成式 AI](#) 技術，其中 [LLM](#) 會在產生回應之前參考訓練資料來源以外的授權資料來源。例如，RAG 模型可能會對組織的知識庫或自訂資料執行語意搜尋。如需詳細資訊，請參閱[什麼是 RAG](#)。

輪換

定期更新[秘密](#)的程序，讓攻擊者更難存取登入資料。

資料列和資料欄存取控制 (RCAC)

使用已定義存取規則的基本、彈性 SQL 表達式。RCAC 包含資料列許可和資料欄遮罩。

RPO

請參閱[復原點目標](#)。

RTO

請參閱[復原時間目標](#)。

執行手冊

執行特定任務所需的一組手動或自動程序。這些通常是為了簡化重複性操作或錯誤率較高的程序而建置。

S

SAML 2.0

許多身分提供者 (IdP) 使用的開放標準。此功能會啟用聯合單一登入 (SSO)，讓使用者可以登入 AWS 管理主控台 或呼叫 AWS API 操作，而不必為您組織中的每個人在 IAM 中建立使用者。如需有關以 SAML 2.0 為基礎的聯合詳細資訊，請參閱 IAM 文件中的[關於以 SAML 2.0 為基礎的聯合](#)。

SCADA

請參閱[監督控制和資料擷取](#)。

SCP

請參閱[服務控制政策](#)。

秘密

您以加密形式存放的 AWS Secrets Manager 機密或限制資訊，例如密碼或使用者登入資料。它由秘密值及其中繼資料組成。秘密值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱 [Secrets Manager 文件中的 Secrets Manager 秘密中的什麼內容？](#)。

依設計的安全性

透過整個開發程序將安全性納入考量的系統工程方法。

安全控制

一種技術或管理防護機制，它可預防、偵測或降低威脅行為者利用安全漏洞的能力。安全控制有四種主要類型：[預防性](#)、[偵測性](#)、[回應性](#)和[主動性](#)。

安全強化

減少受攻擊面以使其更能抵抗攻擊的過程。這可能包括一些動作，例如移除不再需要的資源、實作授予最低權限的安全最佳實務、或停用組態檔案中不必要的功能。

安全資訊與事件管理 (SIEM) 系統

結合安全資訊管理 (SIM) 和安全事件管理 (SEM) 系統的工具與服務。SIEM 系統會收集、監控和分析來自伺服器、網路、裝置和其他來源的資料，以偵測威脅和安全漏洞，並產生提醒。

安全回應自動化

預先定義和程式設計的動作，旨在自動回應或修復安全事件。這些自動化可做為[偵測或回應](#)式安全控制，協助您實作 AWS 安全最佳實務。自動化回應動作的範例包括修改 VPC 安全群組、修補 Amazon EC2 執行個體或輪換登入資料。

伺服器端加密

由接收資料的 AWS 服務 在其目的地加密資料。

服務控制政策 (SCP)

為 AWS Organizations 中的組織的所有帳戶提供集中控制許可的政策。SCP 會定義防護機制或設定管理員可委派給使用者或角色的動作限制。您可以使用 SCP 作為允許清單或拒絕清單，以指定允許或禁止哪些服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的[服務控制政策](#)。

服務端點

的進入點 URL AWS 服務。您可以使用端點，透過程式設計方式連接至目標服務。如需詳細資訊，請參閱 AWS 一般參考 中的 [AWS 服務 端點](#)。

服務水準協議 (SLA)

一份協議，闡明 IT 團隊承諾向客戶提供的服務，例如服務正常執行時間和效能。

服務層級指標 (SLI)

服務效能方面的測量，例如其錯誤率、可用性或輸送量。

服務層級目標 (SLO)

代表服務運作狀態的目標指標，由[服務層級指標](#)測量。

共同責任模式

描述您與共同 AWS 承擔雲端安全與合規責任的模型。AWS 負責雲端的安全，而負責雲端的安全。如需詳細資訊，請參閱[共同責任模式](#)。

SIEM

請參閱[安全資訊和事件管理系統](#)。

單一故障點 (SPOF)

應用程式的單一關鍵元件故障，可能會中斷系統。

SLA

請參閱[服務層級協議](#)。

SLI

請參閱[服務層級指標](#)。

SLO

請參閱[服務層級目標](#)。

先拆分後播種模型

擴展和加速現代化專案的模式。定義新功能和產品版本時，核心團隊會進行拆分以建立新的產品團隊。這有助於擴展組織的能力和服務，提高開發人員生產力，並支援快速創新。如需詳細資訊，請參閱[中的階段式應用程式現代化方法 AWS 雲端](#)。

SPOF

請參閱[單一故障點](#)。

星狀結構描述

使用一個大型事實資料表來存放交易或測量資料的資料庫組織結構，並使用一或多個較小的維度資料表來存放資料屬性。此結構旨在用於[資料倉儲](#)或商業智慧用途。

Strangler Fig 模式

一種現代化單一系統的方法，它會逐步重寫和取代系統功能，直到舊式系統停止使用為止。此模式源自無花果藤，它長成一棵馴化樹並最終戰勝且取代了其宿主。該模式由[Martin Fowler 引入](#)，作為重寫單一系統時管理風險的方式。如需有關如何套用此模式的範例，請參閱[使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET \(ASMX\) Web 服務](#)。

子網

您 VPC 中的 IP 地址範圍。子網必須位於單一可用區域。

監控控制和資料擷取 (SCADA)

在製造中，使用硬體和軟體來監控實體資產和生產操作的系統。

對稱加密

使用相同金鑰來加密及解密資料的加密演算法。

合成測試

以模擬使用者互動的方式測試系統，以偵測潛在問題或監控效能。您可以使用 [Amazon CloudWatch Synthetics](#) 來建立這些測試。

系統提示

一種向 [LLM](#) 提供內容、指示或指導方針以指示其行為的技術。系統提示有助於設定內容，並建立與使用者互動的規則。

T

標籤

做為中繼資料以組織 AWS 資源的鍵值對。標籤可協助您管理、識別、組織、搜尋及篩選資源。如需詳細資訊，請參閱 [標記您的 AWS 資源](#)。

目標變數

您嘗試在受監督的 ML 中預測的值。這也被稱為結果變數。例如，在製造設定中，目標變數可能是產品瑕疵。

任務清單

用於透過執行手冊追蹤進度的工具。任務清單包含執行手冊的概觀以及要完成的一般任務清單。對於每個一般任務，它包括所需的預估時間量、擁有者和進度。

測試環境

請參閱 [環境](#)。

訓練

為 ML 模型提供資料以供學習。訓練資料必須包含正確答案。學習演算法會在訓練資料中尋找將輸入資料屬性映射至目標的模式 (您想要預測的答案)。它會輸出擷取這些模式的 ML 模型。可以使用 ML 模型，來預測您不知道的目標新資料。

傳輸閘道

可以用於互連 VPC 和內部部署網路的網路傳輸中樞。如需詳細資訊，請參閱 AWS Transit Gateway 文件中的 [什麼是傳輸閘道](#)。

主幹型工作流程

這是一種方法，開發人員可在功能分支中本地建置和測試功能，然後將這些變更合併到主要分支中。然後，主要分支會依序建置到開發環境、生產前環境和生產環境中。

受信任的存取權

將許可授予您指定的服務，以代表您在組織中 AWS Organizations 及其帳戶中執行任務。受信任的服務會在需要該角色時，在每個帳戶中建立服務連結角色，以便為您執行管理工作。如需詳細資訊，請參閱文件中的 AWS Organizations [搭配使用 AWS Organizations 與其他 AWS 服務](#)。

調校

變更訓練程序的各個層面，以提高 ML 模型的準確性。例如，可以透過產生標籤集、新增標籤、然後在不同的設定下多次重複這些步驟來訓練 ML 模型，以優化模型。

雙比薩團隊

兩個比薩就能吃飽的小型 DevOps 團隊。雙披薩團隊規模可確保軟體開發中的最佳協作。

U

不確定性

這是一個概念，指的是不精確、不完整或未知的資訊，其可能會破壞預測性 ML 模型的可靠性。有兩種類型的不確定性：認知不確定性是由有限的、不完整的資料引起的，而隨機不確定性是由資料中固有的噪聲和隨機性引起的。如需詳細資訊，請參閱[量化深度學習系統的不確定性指南](#)。

未區分的任務

也稱為繁重工作，這是建立和操作應用程式的必要工作，但不為最終使用者提供直接價值或提供競爭優勢。未區分任務的範例包括採購、維護和容量規劃。

較高的環境

請參閱 [環境](#)。

V

清空

一種資料庫維護操作，涉及增量更新後的清理工作，以回收儲存並提升效能。

版本控制

追蹤變更的程序和工具，例如儲存庫中原始程式碼的變更。

VPC 對等互連

兩個 VPC 之間的連線，可讓您使用私有 IP 地址路由流量。如需詳細資訊，請參閱 Amazon VPC 文件中的[什麼是 VPC 對等互連](#)。

漏洞

危及系統安全性的軟體或硬體瑕疵。

W

暖快取

包含經常存取的目前相關資料的緩衝快取。資料庫執行個體可以從緩衝快取讀取，這比從主記憶體或磁碟讀取更快。

暖資料

不常存取的資料。查詢這類資料時，通常可接受中等緩慢的查詢。

視窗函數

SQL 函數，對與目前記錄在某種程度上相關的資料列群組執行計算。視窗函數適用於處理任務，例如根據目前資料列的相對位置計算移動平均值或存取資料列的值。

工作負載

提供商業價值的資源和程式碼集合，例如面向客戶的應用程式或後端流程。

工作串流

遷移專案中負責一組特定任務的功能群組。每個工作串流都是獨立的，但支援專案中的其他工作串流。例如，組合工作串流負責排定應用程式、波次規劃和收集遷移中繼資料的優先順序。組合工作串流將這些資產交付至遷移工作串流，然後再遷移伺服器和應用程式。

WORM

請參閱[寫入一次，多次讀取](#)。

WQF

請參閱[AWS 工作負載資格架構](#)。

寫入一次，讀取許多 (WORM)

儲存模型，可一次性寫入資料，並防止刪除或修改資料。授權使用者可以視需要多次讀取資料，但無法變更資料。此資料儲存基礎設施被視為[不可變](#)。

Z

零時差入侵

利用[零時差漏洞](#)的攻擊，通常是惡意軟體。

零時差漏洞

生產系統中未緩解的缺陷或漏洞。威脅行為者可以使用這種類型的漏洞來攻擊系統。開發人員經常因為攻擊而意識到漏洞。

零鏡頭提示

提供 [LLM](#) 執行任務的指示，但沒有可協助引導任務的範例 (快照)。LLM 必須使用其預先訓練的知識來處理任務。零鏡頭提示的有效性取決於任務的複雜性和提示的品質。另請參閱[少量擷取提示](#)。

殭屍應用程式

CPU 和記憶體平均使用率低於 5% 的應用程式。在遷移專案中，通常會淘汰這些應用程式。

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。