



使用 AWS CDK in TypeScript 來創建 IaC 項目的最佳實踐

AWS 規定指引



AWS 規定指引: 使用 AWS CDK in TypeScript 來創建 IaC 項目的最佳實踐

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，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，或由 Amazon 贊助。

Table of Contents

簡介	1
目標	2
最佳實務	3
組織大型專案的程式碼	3
為什麼程式碼組織很重要	3
如何組織您的程式碼以進行擴展	3
程式碼組織範例	4
開發可重複使用的模式	5
抽象工廠	5
責任鏈	6
建立或延伸建構模組	7
什麼是建構模組	7
不同類型的建構模組是什麼	7
如何建立自己的建構模組	8
建立或延伸 L2 建構模組	8
建立 L3 建構模組	9
逃生艙	10
自訂資源	11
遵循 TypeScript 最佳做法	14
描述您的資料	14
使用列舉	14
使用介面	15
延伸介面	16
避免空介面	16
使用工廠	17
對屬性使用解構	17
定義標準命名慣例	17
不要使用 var 關鍵字	17
考慮使用 ESLint 和 Prettier	18
使用 access 修飾詞	18
使用公用程式型	18
掃描安全漏洞和格式錯誤	19
安全方法和工具	20
常見開發工具	20

開發和完善文件	21
為什麼 AWS CDK 構造需要代碼文檔	21
TypeDoc 搭配建 AWS 構程式庫使用	21
採用測試驅動的開發方法	22
單元測試	23
集成測試	25
對建構模組使用發行版本和版本控制	25
版本控制 AWS CDK	25
AWS CDK 建構的儲存庫和封裝	25
的建構釋放 AWS CDK	26
強制執行程式庫版本管理	27
常見問答集	29
什麼問題可以 TypeScript 解決？	29
我為什麼要使用 TypeScript？	29
我應該使用AWS CDK或 CloudFormation？	29
如果AWS CDK不支持新推出的怎麼AWS 服務辦？	29
支持哪些不同的編程語言AWS CDK？	29
費用是多AWS CDK少？	29
後續步驟	30
資源	31
文件歷史紀錄	32
詞彙表	33
#	33
A	33
B	36
C	37
D	40
E	43
F	45
G	46
H	47
I	48
L	50
M	50
O	54
P	56

Q	58
R	58
S	61
T	64
U	65
V	65
W	66
Z	67
.....	lxviii

在 TypeScript 中使用 AWS CDK 建立 IaC 專案的最佳實務

Sandeep Gawande、Mason Cahill、Sandip Gangapadhyay、Siamak Heshmati 和 Rajneesh Tyagi，Amazon Web Services (AWS)

2024 年 2 月 ([文件歷史記錄](#))

本指南提供使用 [AWS Cloud Development Kit \(AWS CDK\)](#) in TypeScript 來建置和部署大型基礎結構即程式碼 (IaC) 專案的建議和最佳作法。這 AWS CDK 是一個框架，用於在代碼中定義雲基礎架構，並通過 AWS CloudFormation。如果您沒有明確定義的項目結構，則為大型項目構建和管理 AWS CDK 代碼庫可能很具挑戰性。為了應對這些挑戰，一些組織對大型專案使用反面模式，但這些模式可能會減慢您的專案速度並產生其他問題，對您的組織產生負面影響。例如，反面模式可能會使開發人員登入、錯誤修正和新功能的採用變得複雜並減慢速度。

本指南提供了使用反面模式的替代方案，並顯示如何組織程式碼以實現可擴展性、測試以及與安全最佳實務的一致性。您可以使用本指南來提高 IaC 專案的程式碼品質並最大限度地提高業務敏捷性。本指南適用於建築師、技術主管、基礎設施工程師，以及任何其他尋求為大型專案建立架構良好 AWS CDK 專案的角色。

目標

本指南可以協助您實現下列目標業務成果：

- 降低成本 — 您可以使用設計自己的可重複使用元件，以符合組織的安全性、合規性和治理需求。AWS CDK您也可以輕鬆地在組織中共用元件，以便快速啟動預設符合最佳實務的新專案。
- 加快產品上市時間 — 利用中熟悉的功能AWS CDK來加速您的開發流程。這增加了部署的可重用性並減少開發工作。
- 提高開發人員的生產力 — 開發人員可以使用熟悉的程式設計語言來 這有助於開發人員表達和維護 AWS資源。這可以提高開發人員的效率和協作。

最佳實務

本節將概要介紹下列最佳實務：

- [組織大型專案的程式碼](#)
- [開發可重複使用的模式](#)
- [建立或延伸建構模組](#)
- [遵循 TypeScript 最佳做法](#)
- [掃描安全漏洞和格式錯誤](#)
- [開發和完善文件](#)
- [採用測試驅動的開發方法](#)
- [對建構模組使用發行版本和版本控制](#)
- [強制執行程式庫版本管理](#)

組織大型專案的程式碼

為什麼程式碼組織很重要

對於大型 AWS CDK 項目而言，擁有高質量，定義良好的結構至關重要。隨著專案變得越來越大，支援的功能和建構模組數量增加，程式碼導覽變得更加困難。此困難可能會影響生產力並減慢開發人員登入速度。

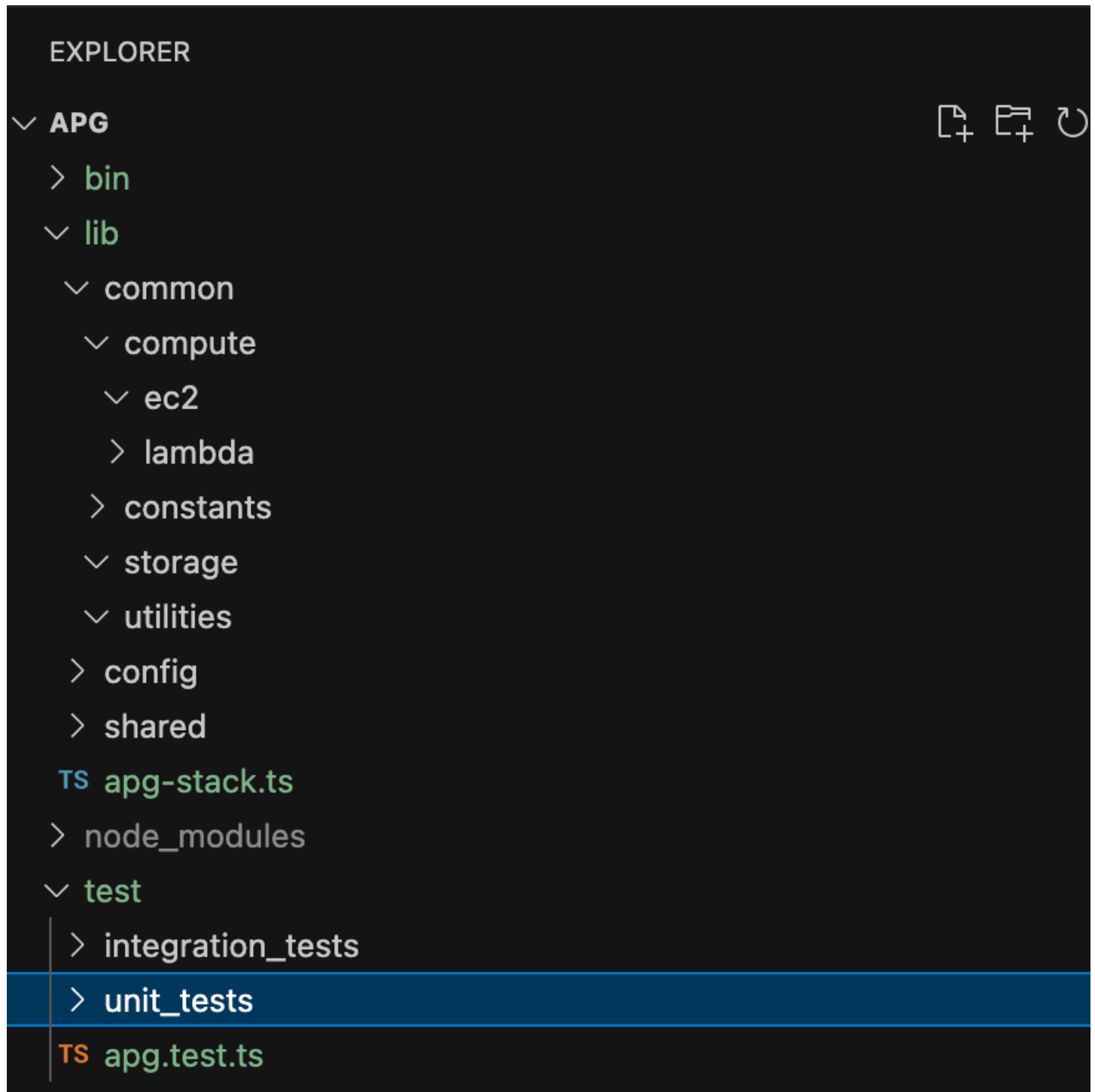
如何組織您的程式碼以進行擴展

為了實現高水準的程式碼靈活性和可讀性，我們建議您根據功能將程式碼劃分為邏輯部分。此劃分反映了這樣一個事實：您的大多數建構模組都用於不同的業務領域。例如，您的前端和後端應用程序都可能需要一個 AWS Lambda 函數並使用相同的源代碼。工廠可以在不向用戶端暴露建立邏輯的情況下建立物件，並使用通用介面來引用新建立的物件。您可以使用工廠作為在程式碼庫中建立一致行為的有效模式。此外，工廠可以作為單一事實來源，協助您避免重複程式碼並使疑難排解變得更加容易。

為了更好地了解工廠的運作方式，請考慮汽車製造商的範例。汽車製造商不需要具備製造輪胎所需的知識和基礎設施。相反，汽車製造商將該專業知識外包給專業的輪胎製造商，然後只需根據需要從該製造商訂購輪胎即可。相同的原則也適用於程式碼。例如，您可以建立一個能夠建置高品質 Lambda 函數的 Lambda 工廠，然後在需要建立 Lambda 函數時在程式碼中呼叫該 Lambda 工廠。同樣，您可以使用此相同的外包程序來解耦應用程式並建置模組化元件。

程式碼組織範例

下面的示 TypeScript 例項目，如下圖所示，包括一個共同的文件夾，你可以保持所有的結構或共同的功能。



例如，compute 資料夾 (駐留在 common 資料夾中) 保存不同運算建構模組的所有邏輯。新開發人員可以輕鬆新增運算建構模組，而不會影響其他資源。所有其他構造都不需要在內部創建新資源。相反，這些建構模組只需呼叫通用建構模組工廠。您可以用相同的方式組織其他建構模組，例如儲存。

組態包含環境型資料，您必須從保存邏輯的 common 資料夾解耦。我們建議您將常用 config 資料放在共用資料夾中。我們還建議您使用 utilities 資料夾來提供所有協助程式功能並清理指令碼。

開發可重複使用的模式

軟體設計模式是軟體開發中常見問題的可重複使用解決方案。其作為指南或範例，可協助軟體工程師建立遵循最佳實務的產品。本節概述了您可以在 AWS CDK 代碼庫中使用的兩種可重複使用模式的概述：抽象工廠模式和責任鏈模式。您可以使用每個模式作為藍圖，並針對程式碼中的特定設計問題進行自訂。如需有關設計模式的詳細資訊，請參閱 Refactoring.Guru 文件中的[設計模式](#)。

抽象工廠

「抽象工廠」模式提供了用於建立相關或相依物件的系列的介面，而無需指定其具體類別。此模式適用於下列使用案例：

- 當用戶端獨立於您在系統中建立和撰寫物件的方式時
- 當系統由多個物件系列組成，且這些系列設計為一起使用時
- 當您必須具有執行期值來建構特定相依性時

有關抽象工廠模式的更多信息，請參閱重構 .Guru 文檔 [TypeScript 中的抽象工廠](#)。

下列程式碼範例顯示如何使用「抽象工廠」模式建構 Amazon Elastic Block Store (Amazon EBS) 儲存工廠。

```
abstract class EBSStorage {
    abstract initialize(): void;
}

class ProductEbs extends EBSStorage {
    constructor(value: String) {
        super();
        console.log(value);
    }
    initialize(): void {}
}
```

```
abstract class AbstractFactory {
    abstract createEbs(): EBSStorage
}

class EbsFactory extends AbstractFactory {
    createEbs(): ProductEbs{
        return new ProductEbs('EBS Created.')
    }
}

const ebs = new EbsFactory();
ebs.createEbs();
```

責任鏈

責任鏈是一種行為設計模式，可讓您沿著潛在處理常式鏈傳遞請求，直到其中一個處理常式處理該請求。「責任鏈」模式適用於下列使用案例：

- 當在執行期確定的多個物件是處理請求的候選者時
- 當您不想在代碼中明確指定處理程序時
- 當您想要向多個物件之一發出請求而不明確指定接收者時

如需有關責任鏈模式的詳細資訊，請參閱 Refactoring.Guru 文件 [TypeScript中的「責任鏈」](#)。

下列程式碼顯示如何使用「責任鏈」模式建置完成任務所需的一系列動作的範例。

```
interface Handler {
    setNext(handler: Handler): Handler;
    handle(request: string): string;
}

abstract class AbstractHandler implements Handler
{
    private nextHandler: Handler;
    public setNext(handler: Handler): Handler {
        this.nextHandler = handler;
        return handler;
    }

    public handle(request: string): string {
        if (this.nextHandler) {
            return this.nextHandler.handle(request);
        }
    }
}
```

```
    }
    return '';
  }
}

class KMSHandler extends AbstractHandler {
  public handle(request: string): string {
    return super.handle(request);
  }
}
```

建立或延伸建構模組

什麼是建構模組

構造是 AWS CDK 應用程式的基本構建塊。建構可以代表單一 AWS 資源，例如 Amazon Simple Storage Service (Amazon S3) 儲存貯體，也可以是由多個 AWS 相關資源組成的更高層級抽象。建構模組的元件可以包括具有關聯運算容量的工作者佇列，或具有監控資源和儀表板的排定作業。AWS CDK 包括稱為「建構程式庫」的結 AWS 構集合。該庫包含每 AWS 服務個。您可以使用「[建構中心](#)」來探索來自 AWS 第三方和開放原始碼 AWS CDK 社群的其他建構。

不同類型的建構模組是什麼

有三種不同類型的建構：AWS CDK

- L1 構造-第 1 層或 L1，構造完全是由定義的資源 CloudFormation-不多，不少。您必須自行提供組態所需的資源。這些 L1 結構是非常基本的，必須手動配置。L1 建構具有Cfn前置詞，並且直接對應於規格。CloudFormation 只 AWS CDK 要 AWS 服務 有這些服務的支援，CloudFormation 就會支援新功能。[CfnBucket](#)是 L1 構造的一個很好的例子。此類別代表 S3 儲存貯體，您必須在其中明確設定所有屬性。我們建議您只有在找不到 L2 或 L3 建構時才使用 L1 建構。
- L2 建構模組 – 第 2 層即 L2 建構模組具有通用樣板程式碼和 glue 邏輯。這些結構具有方便的默認值，並減少您需要了解它們的知識量。L2 構造使用基於意向的 API 來構建您的資源，並通常封裝其對應的 L1 模塊。L2 建構模組的一個良好範例是[儲存貯體](#)。此類別會使用預設屬性和方法 (例如 [bucket.add Lifecycle Rule \(\)](#) 建立 S3 儲存貯體，該儲存貯體會將生命週期規則新增至儲存貯體。
- L3 建構模組 – 第 3 層即 L3 建構模組稱為模式。L3 建構旨在協助您完成中的一般工作 AWS，通常涉及多種資源。這些比 L2 建構模組更加具體、更固定，並且為特定使用案例提供服務。例如，在 [AWS-EC-模式](#)。[ApplicationLoadBalancedFargate服務](#)建構代表包含使用應用 Application Load Balancer 器之 AWS Fargate 容器叢集的架構。另一個例子是 [aws-apigateway](#)。[LambdaRestAPI](#) 的構造。此建構模組代表 Lambda 函數支援的 Amazon API Gateway API。

隨著建構模組層級變得更高，對如何使用這些建構模組進行了更多假設。這可讓您為高度特定的使用案例提供具有更有效預設值的介面。

如何建立自己的建構模組

若要定義自己的建構模組，您必須遵循特定的方法。這是因為所有建構模組都延伸 `Construct` 類別。`Construct` 類別是建構模組樹狀目錄的建構區塊。建構模組在延伸 `Construct` 基本類別的類別中實作。所有建構模組在初始化時都採用以下三個參數：

- 範圍 — 建構的父項或擁有者 (堆疊或其他建構)，決定其在建構樹狀結構中的位置。通常必須為範圍傳遞 `this` (或 Python 中的 `self`)，它表示目前物件。
- `id` — 在此範圍內必須是唯一的識別符。此識別碼可做為目前建構內所定義之所有項目的命名空間，用來配置唯一身分識別，例如資源名稱和 CloudFormation 邏輯 ID。
- `Prop` — 定義建構初始組態的一組屬性。

下列範例顯示如何設定建構模組。

```
import { Construct } from 'constructs';

export interface CustomProps {
  // List all the properties
  Name: string;
}

export class MyConstruct extends Construct {
  constructor(scope: Construct, id: string, props: CustomProps) {
    super(scope, id);

    // TODO
  }
}
```

建立或延伸 L2 建構模組

L2 結構表示「雲端元件」，並封裝 CloudFormation 必須建立元件的所有項目。L2 建構可以包含一或多個 AWS 資源，而且您可以自行自行訂建構。建立或擴充 L2 建構的優點在於，您可以重複使用 CloudFormation 堆疊中的元件，而不需要重新定義程式碼。您可以簡單地將建構模組匯入為類別。

當與現有建構有「是」關係時，您可以延伸現有建構來新增其他預設特徵。重複使用現有 L2 建構的屬性是最佳作法。您可以直接在建構函數中修改屬性來覆寫屬性。

下列範例顯示如何與最佳實務保持一致並延伸稱為 `s3.Bucket` 的現有 L2 建構模組。此延伸建立預設屬性 (例如 `versioned`、`publicReadAccess`、`blockPublicAccess`)，以確保從此新的建構模組建立的所有物件 (在本範例中為 S3 儲存貯體) 永遠設定這些預設值。

```
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
export class MySecureBucket extends s3.Bucket {
  constructor(scope: Construct, id: string, props?: s3.BucketProps) {

    super(scope, id, {
      ...props,
      versioned: true,
      publicReadAccess: false,
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL
    });
  }
}
```

建立 L3 建構模組

當存在與現有構造組合「具有」關係時，組合是更好的選擇。組合意味著您在其他現有建構模組之上建置自己的建構模組。您可以建立自己的模式，將所有資源及其預設值封裝在單一可共用的更高層級的 L3 建構模組內。建立自己的 L3 建構模組 (模式) 的好處是您可以重複使用堆疊中的元件，而無需重新定義程式碼。您可以簡單地將建構模組匯入為類別。這些模式旨在協助取用者以簡潔的方式利用有限的知識，並基於通用模式提供多種資源。

下列程式碼範例會建立名為的 AWS CDK 建構 `ExampleConstruct`。您可以使用此建構模組作為範本定義雲端元件。

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

export interface ExampleConstructProps {
  //insert properties you wish to expose
}

export class ExampleConstruct extends Construct {
  constructor(scope: Construct, id: string, props: ExampleConstructProps) {
    super(scope, id);
    //Insert the AWS components you wish to integrate
  }
}
```

```
}
```

下列範例會示範如何在 AWS CDK 應用程式或堆疊中匯入新建立的建構。

```
import { ExampleConstruct } from './lib/construct-name';
```

下列範例顯示如何執行個體化從基本類別延伸的建構模組的執行個體。

```
import { ExampleConstruct } from './lib/construct-name';

new ExampleConstruct(this, 'newConstruct', {
  //insert props which you exposed in the interface `ExampleConstructProps`
});
```

如需詳細資訊，請參閱[AWS CDK 工作坊](#)文件中的 AWS CDK 工作坊。

逃生艙

您可以在中使用逸出填充線 AWS CDK 來上升抽象層級，以便您可以存取較低層級的建構。逸出剖面線用於延伸特徵的建構，這些特徵未以目前版本的顯示，AWS 但在中 CloudFormation 可用。

我們建議您在下列案例中使用逃生艙：

- AWS 服務 特徵可透過使用 CloudFormation，但沒有其 Construct 建構。
- 一個 AWS 服務 功能可以通過使用，CloudFormation 並且有服務的 Construct 構造，但這些還沒有公開該功能。因為 Construct 建構是「手動開發的」，所以它們有時可能會落後於資 CloudFormation 源結構。

下列範例程式碼顯示使用逃生艙的常見使用案例。在此範例中，較高層級建構模組中尚未實作的功能是新增 httpPutResponseHopLimit 以自動擴展 LaunchConfiguration。

```
const launchConfig = autoscaling.onDemandASG.node.findChild("LaunchConfig") as
  CfnLaunchConfiguration;
  launchConfig.metadataOptions = {
    httpPutResponseHopLimit: autoscalingConfig.httpPutResponseHopLimit ||
2
  }
```

上述程式碼範例顯示下列工作流程：

1. 您可以使用 L2 建構模組來定義 `AutoScalingGroup`。L2 建構不支援更新 `httpPutResponseHopLimit`，因此您必須使用逸出剖面線。
2. 您可以存取 L2 `AutoScalingGroup` 建構模組上的 `node.defaultChild` 屬性並將其轉換為 `CfnLaunchConfiguration` 資源。
3. 現在您可以在 L1 `CfnLaunchConfiguration` 上設定 `launchConfig.metadataOptions` 屬性。

自訂資源

您可以使用自訂資源，在每當您建立、更新 (如果變更自訂資源) 或刪除堆疊時 CloudFormation 執行的範本中撰寫自訂佈建邏輯。例如，如果您想要包含中不可用的資源，可以使用自訂資源 AWS CDK。以此方式，您仍然可以在單一堆疊中管理所有相關資源。

建置自訂資源涉及編寫可回應資源的 CREATE、UPDATE 和 DELETE 生命週期事件的 Lambda 函數。如果您的自訂資源必須只進行單一 API 呼叫，請考慮使用 [AwsCustomResource](#) 建構。這樣可以在 CloudFormation 部署期間執行任意 SDK 呼叫。否則，我們建議您編寫自己的 Lambda 函數來執行您必須完成的工作。

如需有關自訂資源的詳細資訊，請參閱 CloudFormation 文件中的 [自訂資源](#)。有關如何使用自訂資源的範例，請參閱上的 [自訂資源](#) 存放庫 GitHub。

下列範例顯示如何建立自訂資源類別以啟動 Lambda 函數並傳送 CloudFormation 成功或失敗訊號。

```
import cdk = require('aws-cdk-lib');
import customResources = require('aws-cdk-lib/custom-resources');
import lambda = require('aws-cdk-lib/aws-lambda');
import { Construct } from 'constructs';

import fs = require('fs');

export interface MyCustomResourceProps {
  /**
   * Message to echo
   */
  message: string;
}

export class MyCustomResource extends Construct {
  public readonly response: string;
```



```
constructor(scope: Construct, id: string, props: MyCustomResourceProps) {
    super(scope, id);

    const fn = new lambda.SingletonFunction(this, 'Singleton', {
        uuid: 'f7d4f730-4ee1-11e8-9c2d-fa7ae01bbebc',
        code: new lambda.InlineCode(fs.readFileSync('custom-resource-handler.py',
{ encoding: 'utf-8' })),
        handler: 'index.main',
        timeout: cdk.Duration.seconds(300),
        runtime: lambda.Runtime.PYTHON_3_6,
    });

    const provider = new customResources.Provider(this, 'Provider', {
        onEventHandler: fn,
    });

    const resource = new cdk.CustomResource(this, 'Resource', {
        serviceToken: provider.serviceToken,
        properties: props,
    });

    this.response = resource.getAtt('Response').toString();
}
}
```

下列範例顯示自訂資源的主要邏輯。

```
def main(event, context):
    import logging as log
    import cfnresponse
    log.getLogger().setLevel(log.INFO)

    # This needs to change if there are to be multiple resources in the same stack
    physical_id = 'TheOnlyCustomResource'

    try:
        log.info('Input event: %s', event)

        # Check if this is a Create and we're failing Creates
        if event['RequestType'] == 'Create' and
event['ResourceProperties'].get('FailCreate', False):
            raise RuntimeError('Create failure requested')
```

```
# Do the thing
message = event['ResourceProperties']['Message']
attributes = {
    'Response': 'You said "%s"' % message
}

cfnresponse.send(event, context, cfnresponse.SUCCESS, attributes, physical_id)
except Exception as e:
    log.exception(e)
    # cfnresponse's error message is always "see CloudWatch"
    cfnresponse.send(event, context, cfnresponse.FAILED, {}, physical_id)
```

下列範例顯示 AWS CDK 堆疊如何呼叫自訂資源。

```
import cdk = require('aws-cdk-lib');
import { MyCustomResource } from './my-custom-resource';

/**
 * A stack that sets up MyCustomResource and shows how to get an attribute from it
 */
class MyStack extends cdk.Stack {
    constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
        super(scope, id, props);

        const resource = new MyCustomResource(this, 'DemoResource', {
            message: 'CustomResource says hello',
        });

        // Publish the custom resource output
        new cdk.CfnOutput(this, 'ResponseMessage', {
            description: 'The message that came back from the Custom Resource',
            value: resource.response
        });
    }
}

const app = new cdk.App();
new MyStack(app, 'CustomResourceDemoStack');
app.synth();
```

遵循 TypeScript 最佳做法

TypeScript 是一種擴充功能的語言 JavaScript。這是一種強類型和面向對象的語言。您可以使 TypeScript 用指定在程式碼中傳遞的資料類型，並能夠在類型不符時報告錯誤。本節提供 TypeScript 最佳作法的概觀。

描述您的資料

您可以使用 TypeScript 來描述程式碼中物件和函式的形狀。使用 `any` 類型相當於選擇不進行變數的類型檢查。我們建議您避免在程式碼中使用 `any`。請見此處範例。

```
type Result = "success" | "failure"
function verifyResult(result: Result) {
  if (result === "success") {
    console.log("Passed");
  } else {
    console.log("Failed")
  }
}
```

使用列舉

您可以使用列舉來定義一組具名常數並定義可在程式碼庫中重複使用的標準。我們建議您在全域層級匯出一次列舉，然後讓其他類別匯入並使用這些列舉。假設您想要建立一組可能的動作來擷取程式碼庫中的事件。TypeScript 同時提供以數字和字串為基礎的枚舉。下列範例使用列舉。

```
enum EventType {
  Create,
  Delete,
  Update
}

class InfraEvent {
  constructor(event: EventType) {
    if (event === EventType.Create) {
      // Call for other function
      console.log(`Event Captured :${event}`);
    }
  }
}
```

```
let eventSource: EventType = EventType.Create;
const eventExample = new InfraEvent(eventSource)
```

使用介面

介面是類別的合約。如果您建立合約，則您的使用者必須遵守合約。在下列範例中，使用介面來標準化 props 並確保呼叫者在使用此類別時提供預期參數。

```
import { Stack, App } from "aws-cdk-lib";
import { Construct } from "constructs";

interface BucketProps {
  name: string;
  region: string;
  encryption: boolean;
}

class S3Bucket extends Stack {
  constructor(scope: Construct, props: BucketProps) {
    super(scope);
    console.log(props.name);
  }
}

const app = App();
const myS3Bucket = new S3Bucket(app, {
  name: "my-bucket",
  region: "us-east-1",
  encryption: false
})
```

某些屬性只能在首次建立物件時修改。您可以透過在屬性名稱之前放置 `readonly` 來指定這一點，如下列範例所示。

```
interface Position {
  readonly latitude: number;
  readonly longitude: number;
}
```

延伸介面

延伸介面可減少重複，因為您無需在介面之間複製屬性。此外，程式碼的讀取器可以輕鬆理解應用程式中的關係。

```
interface BaseInterface{
  name: string;
}
interface EncryptedVolume extends BaseInterface{
  keyName: string;
}
interface UnencryptedVolume extends BaseInterface {
  tags: string[];
}
```

避免空介面

我們建議您避免空介面，因為其會產生潛在風險。在下列範例中，有一個名為的空白介面 `BucketProps`。 `myS3Bucket1` 和 `myS3Bucket2` 物件都是有效的，但其會遵循不同的標準，因為介面不會強制執行任何合約。下列程式碼將編譯和列印屬性，但這會在您的應用程式中導致不一致。

```
interface BucketProps {}

class S3Bucket implements BucketProps {
  constructor(props: BucketProps){
    console.log(props);
  }
}

const myS3Bucket1 = new S3Bucket({
  name: "my-bucket",
  region: "us-east-1",
  encryption: false,
});

const myS3Bucket2 = new S3Bucket({
  name: "my-bucket",
});
```

使用工廠

在「抽象工廠」模式中，介面負責建立相關物件的工廠，而無需明確指定其類別。例如，您可以建立 Lambda 工廠來建立 Lambda 函數。您不是在建構中建立新的 Lambda 函數，而是將建立程序委派給工廠。有關此設計模式的更多信息，請參閱重構 .Guru 文檔 [TypeScript 中的抽象工廠](#)。

對屬性使用解構

解構，在 ECMAScript 6 (ES6) 中引入，是一 JavaScript 項功能，它使您能夠從數組或對象中提取多個數據片段並將其分配給自己的變量。

```
const object = {
  objname: "obj",
  scope: "this",
};

const oName = object.objname;
const oScop = object.scope;

const { objname, scope } = object;
```

定義標準命名慣例

強制執行命名慣例可以保持程式碼庫的一致性，並減少在考慮如何命名變數時的額外負荷。我們建議下列作法：

- 對變數和函數名稱使用 camelCase。
- 用 PascalCase 於類別名稱和介面名稱。
- 對介面成員使用 camelCase。
- 用 PascalCase 於類型名稱和枚舉名稱。
- 使用 camelCase 命名檔案 (例如，ebsVolumes.tsx 或 storage.tsb)

不要使用 var 關鍵字

該let語句用於在中聲明一個局部變量 TypeScript。它與關鍵字類似，但與var關鍵字相比，在範圍設定var方面有一些限制。在具有 let 的區塊中宣告的變數只能在該區塊內使用。該var關鍵字不能被塊範圍化，這意味著它可以在特定塊之外訪問 (由表示{})，但不能在其定義的函數之外進行訪問。您可以重新聲明和更新var變量。避免使用var關鍵字是最佳做法。

考慮使用 ESLint 和 Prettier

ESLint 可靜態分析您的程式碼以快速發現問題。您可以使用 ESLint 建立一系列聲明 (稱為 lint 規則) 來定義程式碼的外觀或行為方式。ESLint 還提供自動修復程式建議來協助您改進程式碼。最後，您可以使用 ESLint 從共用外掛程式載入 lint 規則。

Prettier 是一個知名的程式碼格式器，支援多種不同的程式設計語言。您可以使用 Prettier 設定程式碼樣式，以便避免手動格式化程式碼。安裝後，您可以更新您的 `package.json` 檔案並執行 `npm run format` 和 `npm run lint` 命令。

下面的例子說明如何啟用 eSLint 和漂亮的格式為您的項目。AWS CDK

```
"scripts": {
  "build": "tsc",
  "watch": "tsc -w",
  "test": "jest",
  "cdk": "cdk",
  "lint": "eslint --ext .js,.ts .",
  "format": "prettier --ignore-path .gitignore --write '**/*.*(js|ts|json)'"
}
```

使用 access 修飾詞

`private` 修飾符僅 TypeScript 限於同一類的可見性。將 `private` 修飾詞新增至屬性或方法時，您可以在相同類別內存取該屬性或方法。

`public` 修飾詞允許從所有位置存取類別屬性和方法。如果您沒有為屬性和方法指定任何訪問修飾符，則默認情況下它們將採用 `public` 修飾符。

`protected` 修飾詞允許在相同類別和子類別內存取類別的屬性和方法。當您希望在 AWS CDK 應用程序中創建子類時，請使用 `protected` 修飾符。

使用公用程式型

中的公用程式類型 TypeScript 是預先定義的類型函數，可對現有類型執行轉換和操作。這有助於您根據現有類型建立新類型。例如，您可以變更或擷取屬性、將屬性設為選用性或必要屬性，或建立不可變的型別版本。透過使用公用程式類型，您可以定義更精確的類型，並在編譯階段 catch 潛在的錯誤。

部分 <Type>

`Partial` 將輸入類型的所有成員標記 `Type` 為可選。此實用程序返回一個代表給定類型的所有子集的類型。以下是 `Partial` 的範例。

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight: number;
}

let partialDog: Partial<Dog> = {};
```

需要 <Type>

`Required` 做相反的 `Partial`。它使輸入類型的所有成員 `Type` 非可選 (換句話說, 是必需的)。以下是 `Required` 的範例。

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight?: number;
}

let dog: Required<Dog> = {
  name: "scruffy",
  age: 5,
  breed: "labrador",
  weight 55 // "Required" forces weight to be defined
};
```

掃描安全漏洞和格式錯誤

基礎設施即程式碼 (IaC) 和自動化對於企業來說至關重要。由於 IaC 如此強大, 您在管理安全風險方面負有重大責任。常見 IaC 安全風險可能包括下列內容:

- 過度寬限 AWS Identity and Access Management (IAM) 權限
- 開放式安全群組

- 未加密的資源
- 存取日誌未開啟

安全方法和工具

我們建議您實作下列安全方法：

- 開發中的漏洞偵測 – 由於開發和分發軟體修補程式的複雜性，修復生產中的漏洞既昂貴又耗時。此外，生產中的漏洞存在被利用的風險。我們建議您對 IaC 資源使用程式碼掃描，以便在發行到生產之前偵測到並修復漏洞。
- 合規性和自動補救 — AWS Config 提供 AWS 受管規則。這些規則可協助您強制執行合規性，並可讓您使用自動[AWS Systems Manager 化](#)來嘗試自動補救。您也可以使用 AWS Config 規則來建立和關聯自訂自動化文件。

常見開發工具

本節介紹的工具可協助您使用自己的自訂規則來延伸內建功能。我們建議您使自訂規則與組織的標準一致。以下是一些要考慮的常見開發工具：

- 使用 cfn-nag 在範本中識別基礎架構安全性問題，例如寬容 IAM 規則或密碼常值。CloudFormation 如需詳細資訊，請參閱 Stelligent 中的 GitHub [cfn-nag](#) 儲存庫。
- 使用 cdk-nag (受 cfn-nag 啟發) 來驗證指定範圍內的建構模組是否符合一組定義的規則。您也可以使用 cdk-nag 進行規則隱藏和合規報告。[cdk-nag 工具藉由延伸中的方面來驗證建構](#)。AWS CDK 如需詳細資訊，請參閱部落格中的[和 cdk-nag 管理應用程式安全性 AWS Cloud Development Kit \(AWS CDK\) 和合規性](#)。AWS DevOps
- 使用開放原始碼工具 Checkov 對您的 IaC 環境執行靜態分析。Checkov 會掃描 Kubernetes、地形或中的基礎架構程式碼，協助識別雲端錯誤設定。CloudFormation 您可以使用 Checkov 取得不同格式的輸出，包括 JSON、JUnit XML 或 CLI。Checkov 可以透過建置顯示動態程式碼相依性的圖表來有效地處理變數。有關更多信息，請參閱布里奇 GitHub [克烏的 Checkov](#) 儲存庫。
- 使用 TFLint 檢查錯誤和已棄用的語法，並協助您強制執行最佳實務。請注意，TFLint 可能無法驗證供應商特定的問題。有關 TFLint 的更多信息，請參閱來自地形畫的 GitHub [TFLint](#) 儲存庫。
- 使用 Amazon 執 CodeWhisperer 行[安全掃描](#)。CodeWhisperer 是一種 AI 驅動的生產力工具，可以實時生成代碼建議。它可以協助您識別安全性問題、遵循最佳做法，並提高程式碼部署的生產力。

開發和完善文件

文件對於專案的成功至關重要。文件不僅解釋了程式碼的工作原理，還可協助開發人員更好地了解應用程式的特性和功能。開發和完善高品質文件可以加強軟體開發程序，維護高品質軟體，並有助於開發人員之間的知識轉移。

文件分為兩個類別：程式碼內的文件和有關程式碼的支援文件。程式碼內的文件採用註解的形式。有關程式碼的支援文件可以是 README 檔案和外部文件。開發人員將文檔視為開銷並不罕見，因為代碼本身很容易理解。對於小型專案來說可能是這樣，但對於涉及多個團隊的大型專案來說，文件至關重要。

對於代碼的作者來說，這是編寫文檔的最佳實踐，因為他們對其功能有很好的了解。開發人員可能會因維護單獨的支援文件而產生額外負荷。為了克服此挑戰，開發人員可以在程式碼中新增註解，並且可以自動擷取這些註解，以便每個版本的程式碼和文件都將保持同步。

有各種不同的工具可協助開發人員從程式碼中擷取註解並為其產生文件。本指南著重於 TypeDoc 作為 AWS CDK 建構的偏好工具。

為什麼 AWS CDK 構造需要代碼文檔

AWS CDK 一般建構由組織中的多個團隊建立，並在不同的團隊之間共用以供使用。良好的文件可協助建構模組庫的取用者輕鬆地整合建構模組並以最輕鬆的方式建置基礎設施。保持所有文件同步是一項艱巨的任務。我們建議您在程式碼中維護文件，這些程式碼將使用程式 TypeDoc 庫擷取。

TypeDoc 搭配建 AWS 構程式庫使用

TypeDoc 是的文件產生器 TypeScript。您可以使用 TypeDoc 來讀取 TypeScript 原始檔案、剖析這些檔案中的註解，然後產生包含程式碼文件的靜態網站。

下面的代碼顯示了如何 TypeDoc 與 AWS 構造庫集成，然後將以下包添加到您的 `package.json` 文件中 `devDependencies`。

```
{
  "devDependencies": {
    "typedoc-plugin-markdown": "^3.11.7",
    "typescript": "~3.9.7"
  },
```

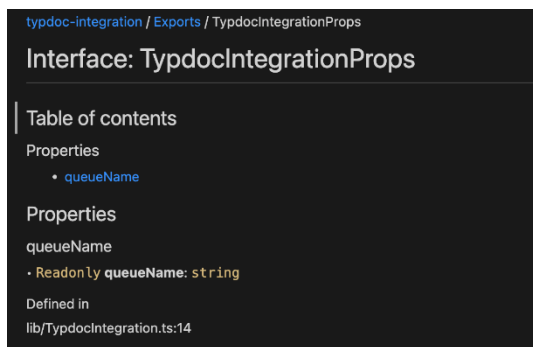
```
}
```

若要將 `typedoc.json` 新增在 CDK 程式庫資料夾中，請使用下列程式碼。

```
{
  "$schema": "https://typedoc.org/schema.json",
  "entryPoints": ["/lib"],
}
```

若要產生 README 檔案，請在 AWS CDK 建構程式庫專案的根目錄中執行 `npx typedoc` 命令。

下列範例文件由產生 TypeDoc。



如需有關 TypeDoc 整合選項的詳細資訊，請參閱[文件中的 TypeDoc 文件註解](#)。

採用測試驅動的開發方法

我們建議您遵循測試驅動開發 (TDD) 方法與 AWS CDK TDD 是一種軟體開發方法，您可以在其中開發測試案例來指定和驗證程式碼。簡言之，首先為每個功能建立測試案例，如果測試失敗，則編寫新程式碼以通過測試並使程式碼簡單且無錯誤。

您可以先使用 TDD 編寫測試案例。這可協助您在強制執行資源安全政策和遵循專案的唯一命名慣例方面驗證具有不同設計限制的基礎設施。測試 AWS CDK 應用程序的標準方法是使用 AWS CDK [斷言](#) 模塊和流行的測試框架，例如 [開玩笑](#) for TypeScript 和 JavaScript 或 [pytest](#) for Python。

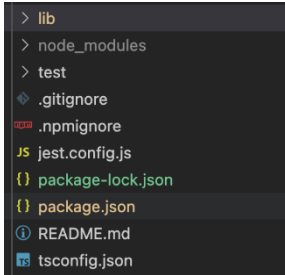
您可以為 AWS CDK 應用程式撰寫兩類測試：

- 使用細粒度的斷言來測試生成的 CloudFormation 模板的特定方面，例如「此資源具有此值的屬性」。這些測試可以偵測迴歸，並且在您使用 TDD 開發新功能時也很有用 (首先編寫測試，然後透過編寫正確的實作使其通過)。精細聲明是您編寫最多的測試。
- 使用快照測試，根據先前儲存的基準 CloudFormation 範本來測試合成的範本。快照測試使自由重構成為可能，因為您可以確保重構的程式碼與原始程式碼的工作方式完全相同。如果變更是有意為之，

您可以接受未來測試的新基準。但是，AWS CDK 升級也可能導致合成模板發生變化，因此您不能僅依賴快照來確保實施正確無誤。

單元測試

本指南 TypeScript 專注於單元測試集成。若要啟用測試，請確保您的 `package.json` 檔案具有下列程式庫：`devDependencies` 中的 `@types/jest`、`jest` 和 `ts-jest`。若要新增這些套件，請執行 `cdk init lib --language=typescript` 命令。在執行上述命令後，您會看到下列結構。



以下代碼是使用 Jest 庫啟用的 `package.json` 文件的示例。

```
{
  ...
  "scripts": {
    "build": "npm run lint && tsc",
    "watch": "tsc -w",
    "test": "jest",
  },
  "devDependencies": {
    ...
    "@types/jest": "27.5.2",
    "jest": "27.5.1",
    "ts-jest": "27.1.5",
    ...
  }
}
```

在 Test 資料夾下，您可以編寫測試案例。下列範例顯示建 AWS CodePipeline 構的測試案例。

```
import {App,Stack} from 'aws-cdk-lib';
import { Template } from 'aws-cdk-lib/assertions';
import * as CodepipelineModule from '../lib/index';
import { Role, ServicePrincipal } from 'aws-cdk-lib/aws-iam';
import { Repository } from 'aws-cdk-lib/aws-codecommit';
```

```
import { PipelineProject } from 'aws-cdk-lib/aws-codebuild';

const testData:CodepipelineModule.CodepipelineModuleProps = {

  pipelineName: "validate-test-pipeline",
  serviceRoleARN: "",
  codeCommitRepositoryARN: "",
  branchName: "master",
  buildStages:[]
}

test('Code Pipeline Created', () => {
  const app = new App();
  const stack = new Stack(app, "TestStack");
  // WHEN
  const serviceRole = new Role(stack, "testRole", { assumedBy: new
ServicePrincipal('codepipeline.amazonaws.com') })
  const codeCommit = new Repository(stack, "testRepo", {
    repositoryName: "validate-codeCommit-repo"
  });
  const codeBuildProject=new PipelineProject(stack,"TestCodeBuildProject",{});
  testData.serviceRoleARN = serviceRole.roleArn;
  testData.codeCommitRepositoryARN = codeCommit.repositoryArn;
  testData["buildStages"].push({
    stageName:"Deploy",
    codeBuildProject:codeBuildProject
  })
  new CodepipelineModule.CodepipelineModule(stack, 'MyTestConstruct', testData);
  // THEN
  const template = Template.fromStack(stack);

  template.hasResourceProperties('AWS::CodePipeline::Pipeline', {
    Name:testData.pipelineName
  });
});
```

若要執行測試，請在專案中執行 `npm run test` 命令。此測試會傳回下列結果。

```
PASS test/codepipeline-module.test.ts (5.972 s)
  # Code Pipeline Created (97 ms)
Test Suites: 1 passed, 1 total
Tests:      1 passed, 1 total
```

```
Snapshots: 0 total
Time:      6.142 s, estimated 9 s
```

有關測試用例的詳細信息，請參閱AWS Cloud Development Kit (AWS CDK) 開發人員指南中的[測試結構](#)。

集成測試

也可以通過使用integ-tests模塊包含 AWS CDK 構造的集成測試。集成測試應該被定義為一個應 AWS CDK 用程序。應該有一個集成測試和應 AWS CDK 用程序之間的 one-to-one 關係。如需詳細資訊，請造訪 AWS CDK API 參考中的[integ-tests-alpha 模組](#)。

對建構模組使用發行版本和版本控制

版本控制 AWS CDK

AWS CDK 通用建構可由多個專案團隊建立，並在組織間共用以供使用。通常，開發人員會在其通用 AWS CDK 結構中發布新功能或錯誤修復。這些結構被 AWS CDK 應用程式或任何其他現有的 AWS CDK 構造用作依賴關係的一部分。因此，開發人員獨立更新和發行具有正確語意版本的建構模組至關重要。下游 AWS CDK 應用程式或其他建 AWS CDK 構可以更新其相依性，以使用新發行的 AWS CDK 建構版本。

語意版本控制 (Semver) 是一組規則或方法，用於為電腦軟體提供唯一的軟體編號。版本定義如下：

- 主要版本包含不相容的 API 變更或重大變更。
- 次要版本包含以回溯相容方式新增的功能。
- 修補版本包含回溯相容的錯誤修正。

如需語意版本化的詳細資訊，請參閱[語意版本化說明文件中的語意版本化規格 \(SemVer\)](#)。

AWS CDK 建構的儲存庫和封裝

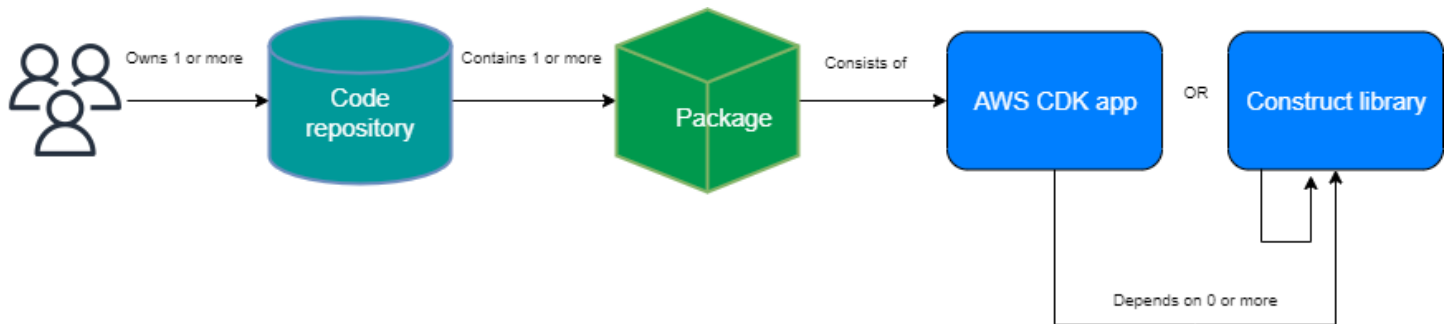
由於 AWS CDK 構造是由不同的團隊開發的，並且由多個 AWS CDK 應用程式使用，因此您可以為每個 AWS CDK 構造使用單獨的存儲庫。這也可以協助您強制執行存取控制。每個存儲庫都可以包含與同一 AWS CDK 構造相關的所有源代碼及其所有依賴關係。透過將單一應用程式 (也就是建 AWS CDK 構) 保留在單一儲存庫中，您可以減少部署期間變更的影響範圍。

它 AWS CDK 不僅會產生用於部署基礎設施的 CloudFormation 範本，還會將 Lambda 函數和 Docker 映像等執行階段資產捆綁在一起，並將它們與您的基礎結構一起部署。不僅可以將定義基礎結構的程式

碼和實作執行階段邏輯的程式碼結合成單一結構，而是最佳作法。這兩種程式碼不需要位於單獨的儲存庫中，甚至不需要位於單獨的套件中。

若要跨儲存庫邊界使用套件，您必須擁有私有套件儲存庫 — 類似於 npm 或 Maven Central PyPi，但是組織內部。您還必須具有發行流程來建置、測試套件並將其發佈至私有套件儲存庫。您可以使用本機虛擬機 PyPi 器 (VM) 或 Amazon S3 來建立私有存放庫，例如伺服器。當您設計或建立私有套件登錄檔時，考慮由於高可用性和可擴展性而導致服務中斷的風險至關重要。託管在雲端以儲存套件的無伺服器受管理服務可大幅降低維護額外負荷。例如，您可以使用 [AWS CodeArtifact](#) 來託管大多數常用程式設計語言的套件。您也可以使用設 CodeArtifact 定外部儲存庫連線，並在其中進行複製 CodeArtifact。

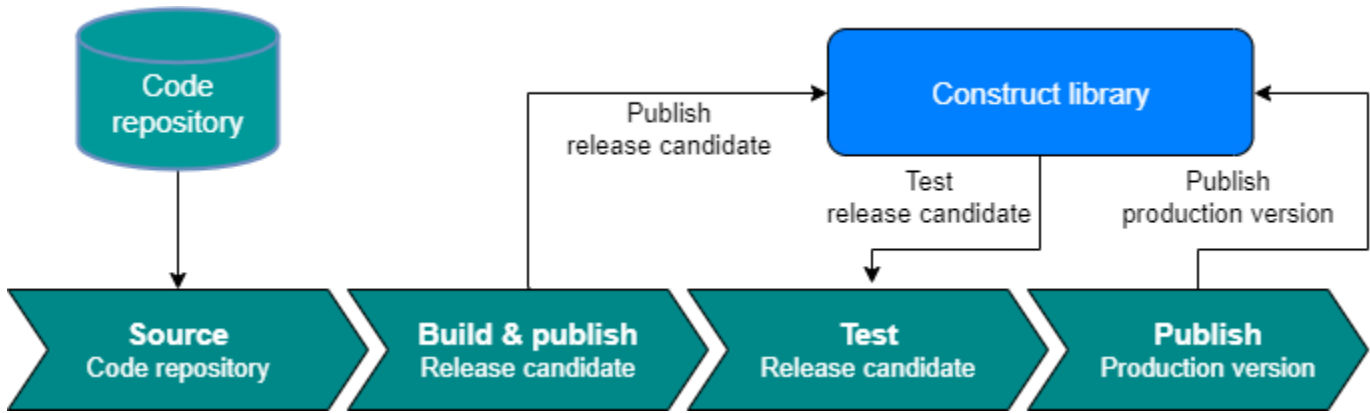
套件儲存庫中套件的相依性是由您語言的套件管理員 (例如 npm for TypeScript 或 JavaScript 應用程式) 來管理。您的套件管理員透過記錄應用程式依賴的每個套件的特定版本來確保建置是可重複的，然後讓您以受控的方式升級這些相依性，如下圖所示。



的建構釋放 AWS CDK

我們建議您建立自己的自動化管道，以建置和發行新的建 AWS CDK 構版本。如果您制定了適當的提取請求核准程序，則一旦您提交並將原始程式碼推送到儲存庫的主要分支中，管道就可以建置並建立發行候選版本。在發布生產就緒版本之前，可以推送該版本 CodeArtifact 並進行測試。或者，您可以在將代碼與主分支合併之前在本地測試新的 AWS CDK 構造版本。這會導致管道發行生產就緒型版本。考慮到共用建構模組和套件必須獨立於使用應用程式進行測試，就像其向公眾發行一樣。

下圖顯示範例 AWS CDK 版本發行管線。



您可以使用下列範例命令來建置、測試和發佈 npm 套件。首先，透過執行下列命令登入成品儲存庫。

```
aws codeartifact login --tool npm --domain <Domain Name> --domain-owner $(aws sts get-caller-identity --output text --query 'Account') \
--repository <Repository Name> --region <AWS Region Name>
```

然後，完成下列步驟：

1. 根據 package.json 文件安裝所需的套件：npm install
2. 建立候選發行版本：npm version prerelease --preid rc
3. 建置 npm 套件：npm run build
4. 測試 npm 套件：npm run test
5. 發佈 npm 套件：npm publish

強制執行程式庫版本管理

當您維護 AWS CDK 程式碼庫時，生命週期管理是一項重大挑戰。例如，假設您使用版本 1.97 啟動 AWS CDK 專案，然後版本 1.169 稍後變為可用。版本 1.169 提供了新功能和錯誤修正，但您已使用舊版部署基礎設施。現在，由於新版本中可能會引入重大變更，因此隨著差距的擴大，更新建構模組變得具有挑戰性。如果您的環境中具有許多資源，這可能是一個挑戰。本節中介紹的模式可以幫助您使用自動化管理 AWS CDK 庫版本。以下是此模式的工作流程：

1. 當您啟動新的 CodeArtifact Service Catalog 產品時，物件 AWS CDK 庫版本及其相依性會儲存在 package.json 檔案中。
2. 您可部署一個通用管道來追蹤所有儲存庫，以便在沒有重大變更時對其套用自動升級。
3. AWS CodeBuild 階段會檢查相依性樹狀結構，並尋找中斷變更。

4. 該管道建立一個功能分支，然後使用新版本執行 `cdk synth` 以確認沒有錯誤。
5. 新版本部署在測試環境中，最後執行整合測試以確保部署運作狀態正常。
6. 您可以使用兩個 Amazon Simple Queue Service (Amazon SQS) 佇列來追蹤堆疊。使用者可以在例外狀況佇列中手動檢閱堆疊並解決重大變更。通過整合測試的項目允許合併和發行。

常見問答集

什麼問題可以 TypeScript 解決？

通常，您可以透過編寫自動化測試，手動驗證程式碼是否如預期般運作，最後讓另一個人驗證您的程式碼來消除程式碼中的錯誤。驗證專案的每個部分之間的連線非常耗時。為了加快驗證過程，您可以使用類型檢查語言，例如 TypeScript 自動化代碼驗證並在開發過程中提供即時反饋。

我為什麼要使用 TypeScript？

TypeScript 是一種簡化 JavaScript 代碼的開源語言，使其更易於閱讀和調試。TypeScript 還為 JavaScript IDE 和實踐（例如靜態檢查）提供了高生產力的開發工具。此外，TypeScript 提供 ECMAScript 6 (ES6) 的優勢，並可以提高您的生產力。最後，TypeScript 可以幫助您避免開發人員 JavaScript 通過類型檢查代碼編寫時經常遇到的痛苦錯誤。

我應該使用 AWS CDK 或 CloudFormation？

如果您的組織具備開發專業知識 AWS CloudFormation，可以利用 AWS CDK。AWS Cloud Development Kit (AWS CDK) 這是因為比更靈活 CloudFormation，因為您可以使用編程語言和 OOP 概念。AWS CDK 請記住，您可以使 CloudFormation 用有序且可預測的方式建立 AWS 資源。在 CloudFormation，資源是使用 JSON 或 YAML 格式以文字檔案撰寫。

如果 AWS CDK 不支持新推出的怎麼 AWS 服務辦？

您可以使用 [原始覆寫](#) 或自 CloudFormation [訂資源](#)。

支持哪些不同的編程語言 AWS CDK？

AWS CDK 通常可在 JavaScript、Python TypeScript、Java、C# 和 Go 中使用 (在開發人員預覽版中)。

費用是多 AWS CDK 少？

沒有任何額外費用 AWS CDK。您需要支付使用時所建立的 AWS 資源 (例如 Amazon EC2 執行個體或 Elastic Load Balancing 負載平衡器) AWS CDK 的費用，方式與手動建立的方式相同。當您使用資源時，僅需為使用的資源付費。沒有最低費用，也不需要前期承諾。

後續步驟

我們建議您開始使用 AWS Cloud Development Kit (AWS CDK) in 進行建置 TypeScript。有關更多信息，請參閱[AWS CDK 沉浸式日研討會](#)。

資源

參考

- [AWS 解決方案建構](#) (解AWS 決方案)
- [AWS 雲端開發套件 \(AWS CDK\)](#) (GitHub)
- [AWS 構建庫 API 參考](#) (AWS CDK 參考文檔)
- [AWS CDK 參考文檔](#) (AWS CDK 參考文檔)
- [AWS CDK 沉浸式日工作坊](#) (AWS 工作坊工作室)

工具

- [CDG-黑客](#) () GitHub
- [TypeScript 網絡鏈接](#) ([電TypeScript子書文檔](#))

指南和模式

- [AWS 解決方案構造模式](#) (AWS 文檔)

文件歷史紀錄

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

變更	描述	日期
更新代碼	我們更新了「 遵循 TypeScript 最佳做法 」一節中的程式碼範例。	2024年2月16日
新增區段	我們新增了 使用公用程式類 和 整合測試 區段。	2024年1月10日
次要更新	已更新用於建立 L3 建構模組的程式碼範例。	2023年6月16日
初次出版	—	2022年12月8日

AWS 規定指引詞彙

以下是 AWS 規範性指引所提供的策略、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：將您的內部部署 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：將您的現場部署 Oracle 資料庫遷移至 Amazon Relational Database Service 服務 (Amazon RDS)，適用於 AWS 雲端中的 Oracle。
- **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：將現場部署 Oracle 資料庫遷移至 AWS 雲端 EC2 執行個體上的 Oracle。
- **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。此移轉案例適用於 VMware Cloud on AWS，支援內部部署環境與之間的虛擬機器 (VM) 相容性和工作負載可攜性 AWS。在將基礎設施遷移至 VMware Cloud on AWS 時，您可以使用內部部署資料中心的 VMware Cloud Foundation 技術。範例：將裝載 Oracle 資料庫的虛擬機器管理程序重新定位至 VMware 雲端。 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

請參閱以[屬性為基礎的存取控制](#)。

抽象的服務

請參閱[受管理服務](#)。

酸

請參閱[原子性、一致性、隔離性、耐用性](#)。

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。它比[主動-被動遷移](#)更具彈性，但需要更多的工作。

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，但只有來源資料庫處理來自連接應用程式的交易，同時將資料複寫至目標資料庫。目標資料庫在遷移期間不接受任何交易。

聚合函數

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

AI

請參閱[人工智慧](#)。

艾奧運

請參閱[人工智慧作業](#)。

匿名化

永久刪除資料集中個人資訊的程序。匿名化可以幫助保護個人隱私。匿名資料不再被視為個人資料。

反模式

一種經常使用的解決方案，用於解決方案的生產力適得其反，效果不佳或效果低於替代方案。

應用控制

一種安全性方法，只允許使用已核准的應用程式，以協助保護系統免受惡意軟體的攻擊。

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 AWS 中的 [ABAC](#)。

授權資料來源

儲存資料主要版本的位置，被認為是最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以便處理或修改資料，例如匿名化、編輯或將其虛擬化。

可用區域

一個獨立的位置，與其他 AWS 區域 可用區域中的故障隔離，並為相同區域中的其他可用區域提供廉價、低延遲的網路連線能力。

AWS 雲端採用架構 (AWS CAF)

指導方針和最佳做法的架構，可協 AWS 助組織制定有效率且有效的計畫，以順利移轉至雲端。AWS CAF 將指導組織到六個重點領域，稱為觀點：業務，人員，治理，平台，安全性和運營。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。針對此觀點，AWS CAF 為人員開發、訓練和通訊提供指導，以協助組織為成功採用雲端做好準備。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)

可評估資料庫移轉工作負載、建議移轉策略並提供工作預估的工具。AWS WQF 包含在 AWS Schema Conversion Tool (AWS SCT) 中。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

壞機器人

旨在破壞或對個人或組織造成傷害的**機器人**。

BCP

請參閱[業務連續性規劃](#)。

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「[位元順序](#)」。

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

建立兩個獨立但相同環境的部署策略。您可以在一個環境中執行目前的應用程式版本 (藍色)，而在另一個環境 (綠色) 中執行新的應用程式版本。此策略可協助您以最小的影響快速回復。

機器人

透過網際網路執行自動化工作並模擬人類活動或互動的軟體應用程式。某些漫遊器是有用的或有益的，例如用於索引 Internet 上信息的網絡爬蟲。其他一些機器人 (稱為不良機器人) 旨在破壞或對個人或組織造成傷害。

殭屍網絡

受[惡意軟件](#)感染並受到單一方（稱為[機器人牧民](#)或[機器人操作員](#)）控制的機器人網絡。殭屍網絡是擴展機器人及其影響的最著名機制。

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)。

防碎玻璃訪問

在特殊情況下，並透過核准的程序，使用者可以快速取得他 AWS 帳戶 們通常沒有存取權限的存取權。如需詳細資訊，請參閱 AWS Well-Architected 指南中的[實作防破玻璃程序](#)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

咖啡

請參閱[AWS 雲端採用架構](#)。

金絲雀部署

向最終用戶發行版本的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前的版本。

CCoE

請參閱[雲端卓越中心](#)。

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來測試系統的彈性。您可以使用 [AWS Fault Injection Service \(AWS FIS\)](#) 執行實驗來 stress 您的 AWS 工作負載並評估其回應。

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。雲計算通常連接到[邊緣計算](#)技術。

雲端運作模式

在 IT 組織中，這是用來建置、成熟和最佳化一或多個雲端環境的作業模型。如需詳細資訊，請參閱[建立您的雲端作業模型](#)。

採用雲端階段

組織遷移到 AWS 雲端時通常會經歷的四個階段：

- 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 Stephen Orban 在雲端企業策略部落格文章 [「邁向 AWS 雲端優先的旅程與採用階段」](#) 部落格文章中定義的。如需其與 AWS 移轉策略之間關聯的詳細資訊，請參閱 [移轉準備指南](#)。

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。常見的雲儲存庫包括 GitHub 或 AWS CodeCommit。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。查詢此類資料時，通常可以接受緩慢的查詢。將此資料移至效能較低且成本較低的儲存層或類別可降低成本。

計算機視覺 (CV)

一個 [AI](#) 領域，它使用機器學習來分析和從數字圖像和視頻等視覺格式中提取信息。例如，提 AWS Panorama 供將 CV 添加到現場部署攝像機網絡的設備，Amazon 為 CV SageMaker 提供圖像處理算法。

配置漂移

對於工作負載，組態會從預期的狀態變更。這可能會導致工作負載變得不合規，而且通常是漸進且無意的。

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

AWS Config 規則和補救動作的集合，您可以組合這些動作來自訂合規性和安全性檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶和區域中的單一實體，或跨組織部署。如需詳細資訊，請參閱文件中的[AWS Config 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 架構中安全性支柱的一個組成部分。如需詳細資訊，請參閱[資料分類](#)。

資料漂移

生產資料與用來訓練 ML 模型的資料之間有意義的變化，或輸入資料隨著時間的推移有意義的變化。資料漂移可降低 ML 模型預測中的整體品質、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

透過集中式管理和控管，提供分散式、分散式資料擁有權的架構架構。

資料最小化

僅收集和處理絕對必要的數據的原則。在中執行資料最小化 AWS 雲端可降低隱私權風險、成本和分析碳足跡。

資料周長

您 AWS 環境中的一組預防性護欄，可協助確保只有受信任的身分正在存取來自預期網路的受信任資源。若要取得更多資訊，請參閱 [〈在上建立資料周長〉](#) 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。

數據來源

在整個生命週期中追蹤資料來源和歷史記錄的程序，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 (例如分析) 的資料管理系統。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。

defense-in-depth

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 結構的不同層加入多個控制項，以協助保護資源。例如，— defense-in-depth 種方法可能會結合多因素驗證、網路分段和加密。

委派的管理員

在中 AWS Organizations，相容的服務可以註冊成 AWS 員帳戶，以管理組織的帳戶並管理該服務的權限。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](#)。

部署

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](#)。

發展價值流映射

用於識別限制並排定優先順序，對軟體開發生命週期中的速度和品質產生不利影響的程序。DVSM 擴展了最初為精益生產實踐而設計的價值流映射流程。它著重於創造和通過軟件開發過程中移動價值所需的步驟和團隊。

數字雙胞胎

真實世界系統的虛擬表現法，例如建築物、工廠、工業設備或生產線。數位雙胞胎支援預測性維護、遠端監控和生產最佳化。

維度表

在 [star 結構描述](#) 中，較小的資料表包含事實資料表中定量資料的相關資料屬性。維度表格屬性通常是文字欄位或離散數字，其行為類似於文字。這些屬性通常用於查詢限制、篩選和結果集標籤。

災難

防止工作負載或系統在其主要部署位置達成其業務目標的事件。這些事件可能是自然災害、技術故障或人為行為造成的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您使用的策略和程序，將因[災難](#)造成的停機時間和資料遺失降到最低。如需詳細資訊，請參閱 AWS Well-Architected [的架構中的雲端中的工作負載的災難復原](#) [AWS：雲端復原](#)。

DML

請參閱[資料庫操作語言](#)。

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](#)。

博士

請參閱[災難復原](#)。

漂移檢測

追蹤基線組態的偏差。例如，您可以用 AWS CloudFormation 來[偵測系統資源中的漂移](#)，也可以用 AWS Control Tower 來[偵測 landing zone 中可能會影響法規遵循治理要求的變更](#)。

DVSM

請參閱[開發價值流映射](#)。

E

EDA

請參閱[探索性資料分析](#)。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與[雲計算](#)相比，邊緣計算可以減少通信延遲並縮短響應時間。

加密

一種計算過程，將純文本數據（這是人類可讀的）轉換為密文。

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。您可以使用其他或 (IAM) 主體建立端點服務，AWS PrivateLink 並將權限授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 主體。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

自動化並管理企業的關鍵業務流程 (例如會計、[MES](#) 和專案管理) 的系統。

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 安全史詩包括身份和訪問管理，偵探控制，基礎結構安全性，數據保護和事件響應。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

事實表

[星型架構](#)中的中央表格。它存儲有關業務運營的定量數據。事實資料表通常包含兩種類型的資料欄：包含計量的資料欄，以及包含維度表格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減少開發生命週期的理念。這是敏捷方法的關鍵組成部分。

故障隔離邊界

在中 AWS 雲端，可用區域、AWS 區域控制平面或資料平面等界限，可限制故障影響並協助改善工作負載的彈性。如需詳細資訊，請參閱[AWS 錯誤隔離邊界](#)。

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[機器學習模型可解釋性：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。

FGAC

請參閱[精細的存取控制](#)。

精細的存取控制 (FGAC)

使用多個條件來允許或拒絕訪問請求。

閃切遷移

一種資料庫移轉方法，透過[變更資料擷取使用連續資料](#)複寫，在最短的時間內移轉資料，而不是使用階段化方法。目標是將停機時間降至最低。

G

地理阻塞

請參閱[地理限制](#)。

地理限制 (地理封鎖)

在 Amazon 中 CloudFront，防止特定國家/地區的使用者存取內容分發的選項。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 工作流程被認為是遺留的，[基於主幹的工作流程是現代的首選方法](#)。

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 許可界限來將其實作。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。它們是通過使用 AWS Config，Amazon AWS Security Hub GuardDuty，AWS Trusted Advisor 亞馬遜檢查 Amazon Inspector 和自定義 AWS Lambda 檢查來實現的。

H

公頃

查看 [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)

工作負載在遇到挑戰或災難時持續運作的能力, 無需干預。HA 系統的設計可自動容錯移轉、持續提供高品質的效能, 以及處理不同的負載和故障, 並將效能影響降到最低。

歷史學家現代化

一種用於現代化和升級操作技術 (OT) 系統的方法, 以更好地滿足製造業的需求。歷史學家是一種類型的數據庫, 用於收集和存儲工廠中的各種來源的數據。

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。

熱數據

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的儲存層或類別, 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, 修補程式通常是在典型的 DevOps 發行工作流程之外進行。

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

查看[基礎結構即程式碼](#)。

身分型政策

附加至一或多個 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。

IIoT

請參閱[工業物聯網](#)。

不可變基礎設施

為生產工作負載部署新基礎結構的模型，而不是更新、修補或修改現有基礎結構。[不可變的基礎架構本質上比可變基礎架構更加一致、可靠且可預測](#)。如需詳細資訊，請參閱 Well-Architected 的架構中的[使用不可變基礎結構 AWS 構進行部署](#)最佳作法。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，VPC 可接受、檢查和路由來自應用程式外部的網路連線。[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

[Klaus Schwab](#) 於 2016 年推出的一個術語，指的是透過連線能力、即時資料、自動化、分析和 AI/ML 的進步來實現製造流程的現代化。

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 VPC (相同或不同 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[AWS 的機器學習模型可解釋性](#)。

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

標籤式存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中每個使用者和資料本身都明確指派一個安全性標籤值。使用者安全性標籤與資料安全性標籤之間的交集決定了使用者可以看到哪些列與欄。

登陸區域

landing zone 是一個架構良好的多帳戶 AWS 環境，具有可擴展性和安全性。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](#)。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱以[標示為基礎的存取控制](#)。

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

見 [7 盧比](#)

小端序系統

首先儲存最低有效位元組的系統。另請參閱 [「位元順序」](#)。

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

旨在危及計算機安全性或隱私的軟件。惡意軟件可能會破壞計算機系統，洩漏敏感信息或獲得未經授權的訪問。惡意軟體的例子包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄程式。

受管理服務

AWS 服務用於 AWS 操作基礎架構層、作業系統和平台，並且您可以存取端點以儲存和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也被稱為抽象的服務。

製造執行系統

用於跟踪，監控，記錄和控制生產過程的軟件系統，可在現場將原材料轉換為成品。

MAP

請參閱 [Migration Acceleration Program](#)。

機制

一個完整的過程，您可以在其中創建工具，推動工具的採用，然後檢查結果以進行調整。機制是一個循環，它加強和改善自己，因為它運行。如需詳細資訊，請參閱 AWS Well-Architected 的架構中[建置機制](#)。

成員帳戶

屬於 AWS 帳戶 中組織的管理帳戶以外的所有帳戶 AWS Organizations。一個帳戶一次只能是一個組織的成員。

MES

請參閱[製造執行系統](#)。

郵件佇列遙測傳輸 (MQTT)

[以發佈/訂閱模式為基礎的輕量型 machine-to-machine \(M2M\) 通訊協定，適用於資源受限 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 AWS 無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[上 AWS 的實作微服務](#)。

Migration Acceleration Program (MAP)

提供諮詢支援、訓練和服務的 AWS 計畫，協助組織為移轉至雲端建立穩固的營運基礎，並協助抵消移轉的初始成本。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。移轉工廠團隊通常包括營運、業務分析師和擁有者、移轉工程師、開發人員和 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。移轉中繼資料的範例包括目標子網路、安全性群組和 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 AWS 用應用程式遷移服務將遷移重新託管到 Amazon EC2。

遷移組合評定 (MPA)

這是一種線上工具，可提供驗證商業案例以移轉至 AWS 雲端的資訊。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)。所有 AWS 顧問和 APN 合作夥伴顧問均可免費使用 [MPA 工具](#) (需要登入)。

遷移準備程度評定 (MRA)

使用 AWS CAF 獲得有關組織雲端準備狀態的見解、識別優勢和弱點，以及建立行動計劃以縮小已識別差距的過程。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

將工作負載移轉至 AWS 雲端時所使用的方法。如需詳細資訊，請參閱本詞彙表中的 [7 Rs](#) 項目，並參閱[動員您的組織以加速大規模移轉](#)。

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[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[評估 AWS 雲端中應用程式的現代化準備情況](#)。

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[移轉組合評估](#)。

MQTT

請參閱[佇列遙測傳輸](#)的郵件。

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？」

可變的基礎

一種模型，用於更新和修改生產工作負載的現有基礎結構。為了提高一致性，可靠性和可預測性，AWS Well-Architected 框架建議使用[不可變的基礎結構](#)作為最佳實踐。

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[作業整合](#)。

OLA

請參閱[作業層級協定](#)。

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。

OPCA

請參閱[開放程序通訊-統一架構](#)。

開放程序通訊-統一架構 (OPC-UA)

用於工業自動化的 machine-to-machine (M2M) 通訊協定。OPC-UA 提供數據加密，身份驗證和授權方案的互操作性標準。

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)。

操作準備程度檢討 (ORR)

問題和相關最佳做法的檢查清單，可協助您瞭解、評估、預防或減少事件和可能的故障範圍。如需詳細資訊，請參閱 AWS Well-Architected 的架構中的[作業準備檢閱 \(ORR\)](#)。

操作技術

可與實體環境搭配使用的硬體和軟體系統，以控制工業作業、設備和基礎設施。在製造業中，整合 OT 和資訊技術 (IT) 系統是[工業 4.0](#) 轉型的關鍵焦點。

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

由建立的追蹤 AWS CloudTrail 記錄中組織 AWS 帳戶 中所有人的所有事件 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文件中的為組織建立追蹤](#)。

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 移轉策略中，這個架構稱為人員加速，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)

在中 CloudFront，限制存取權限以保護 Amazon Simple Storage Service (Amazon S3) 內容的增強選項。OAC 支援所有 S3 儲存貯體 AWS 區域、伺服器端加密 AWS KMS (SSE-KMS)，以及 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

在中 CloudFront，用於限制存取以保護 Amazon S3 內容的選項。當您使用 OAI 時，CloudFront 會建立 Amazon S3 可用來進行驗證的主體。經驗證的主體只能透過特定散發存取 S3 儲存 CloudFront 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[作業整備檢閱](#)。

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的網路連線的 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](#)。

個人識別資訊 (PII)

直接查看或與其他相關數據配對時，可用於合理推斷個人身份的信息。PII 的範例包括姓名、地址和聯絡資訊。

PII

請參閱[個人識別資訊](#)。

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。

公司

請參閱[可編程邏輯控制器](#)

PLM

請參閱[產品生命週期管理](#)

政策

可以定義權限 (請參閱以[身分識別為基礎的策略](#))、指定存取條件 (請參閱以[資源為基礎的策略](#)) 或定義組織中所有帳戶的最大權限的物件 AWS Organizations (請參閱[服務控制策略](#))。

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。如需詳細資訊，請參閱[在微服務中啟用資料持久性](#)。

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，通常位於子WHERE句中。

謂詞下推

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

隱私設計

一種系統工程方法，在整個工程過程中將隱私權納入考量。

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

一種[安全控制項](#)，旨在防止部署不符合規範的資源。這些控制項會在資源佈建之前進行掃描。如果資源不符合控制項，則不會佈建該資源。如需詳細資訊，請參閱 AWS Control Tower 文件中的[控制項參考指南](#)，並參閱實作安全性[控制中的主動](#)控制 AWS。

產品生命週期管理 (PLM)

在產品的整個生命週期中管理資料和流程，從設計、開發、上市到成長與成熟度，再到下降和移除。

生產環境

請參閱[環境](#)。

可編程邏輯控制器 (PLC)

在製造業中，一台高度可靠且適應性強的計算機，可監控機器並自動化製造過程。

化名化

以預留位置值取代資料集中的個人識別碼的程序。假名化可以幫助保護個人隱私。假名化數據仍被認為是個人數據。

發布/訂閱 (發布/訂閱)

一種模式，可在微服務之間實現非同步通訊，以提高延展性和回應能力 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通道。系統可以在不變更發佈服務的情況下新增微服務。

Q

查詢計劃

一系列步驟，如指示，用來存取 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

拉齐矩阵

請參閱[負責任，負責，諮詢，通知 \(RAC I\)](#)。

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。

拉西矩陣

請參閱[負責任，負責，諮詢，通知 \(RACI\)](#)。

RCAC

請參閱[列與欄存取控制](#)。

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。

重新建築師

見 [7 盧比](#)

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這決定了最後一個恢復點和服務中斷之間可接受的數據丟失。

復原時間目標 (RTO)

服務中斷與恢復服務之間的最大可接受延遲。

重構

見 [7 盧比](#)

區域

地理區域中的 AWS 資源集合。每個 AWS 區域 是隔離和獨立於其他的，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 您的帳戶可以使用的項目](#)。

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。

重新主持

見 [7 盧比](#)

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

見 [7 盧比](#)

再平台

見 [7 盧比](#)

買回

見 [7 盧比](#)

彈性

應用程式抵抗或從中斷中復原的能力。在規劃備援時，[高可用性](#)和[災難復原](#)是常見的考量因素。AWS 雲端如需詳細資訊，請參閱[AWS 雲端 復原力](#)。

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

定義移轉活動和雲端作業所涉及之所有各方的角色與責任的矩陣。矩陣名稱衍生自矩陣中定義的責任型別：負責 (R)、負責 (A)、諮詢 (C)，以及通知 (I)。支撐 (S) 類型是可選的。如果您包含支援，則該矩陣稱為 RASCI 矩陣，如果您將其排除，則稱為 R ACI 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

見 [7 盧比](#)

退休

見 [7 盧比](#)

旋轉

定期更新[密碼](#)以使攻擊者更難以存取認證的程序。

資料列與資料行存取控制 (RCAC)

使用已定義存取規則的基本、彈性 SQL 運算式。RCAC 由資料列權限和資料行遮罩所組成。

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

許多身份提供者 (IdPs) 使用的開放標準。此功能可啟用聯合單一登入 (SSO)，因此使用者可以登入 AWS Management Console 或呼叫 AWS API 作業，而不必為組織中的每個人在 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](#)。

斯卡達

請參閱[監督控制和資料擷取](#)。

SCP

請參閱[服務控制策略](#)。

秘密

您以加密形式儲存的機密或受限制資訊，例如密碼或使用者認證。AWS Secrets Manager 它由秘密值及其中繼資料組成。密碼值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱[秘密管理員說明文件](#)中的秘密。

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。安全性控制有四種主要類型：[預防性](#)、[偵測](#)、[回應式](#)和[主動式](#)。

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

預先定義且程式化的動作，其設計用來自動回應或修復安全性事件。這些自動化作業可做為[偵探或回應式](#)安全控制項，協助您實作 AWS 安全性最佳實務。自動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

在其目的地的數據加密，通 AWS 服務 過接收它。

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[服務控制原則](#)。

服務端點

的進入點的 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。

服務等級指示器 (SLI)

對服務效能層面的測量，例如錯誤率、可用性或輸送量。

服務等級目標 (SLO)

代表服務狀況的目標測量結果，由[服務層次指示器](#)測量。

共同責任模式

描述您在雲端安全性和合規方面共享的責任的模型。AWS AWS 負責雲端的安全性，而您則負責雲端的安全性。如需詳細資訊，請參閱[共同責任模式](#)。

暹

請參閱[安全性資訊和事件管理系統](#)。

單點故障 (SPF)

應用程式的單一重要元件發生故障，可能會中斷系統。

SLA

請參閱[服務等級協議](#)。

SLI

請參閱[服務層級指示器](#)。

SLO

請參閱[服務等級目標](#)。

split-and-seed 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[中的應用程式現代化的階段化方法](#)。AWS 雲端

痙攣

請參閱[單一故障點](#)。

星型綱要

使用一個大型事實資料表來儲存交易或測量資料，並使用一或多個較小的維度表格來儲存資料屬性的資料庫組織結構。這種結構是專為在[數據倉庫](#)中使用或用於商業智能目的。

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。

監督控制與資料擷取 (SCADA)

在製造業中，使用硬體與軟體來監控實體資產與生產作業的系統。

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

以模擬使用者互動以偵測潛在問題或監控效能的方式測試系統。您可以使用 [Amazon CloudWatch Synthetics](#) 來創建這些測試。

T

標籤

作為組織 AWS 資源的中繼資料的索引鍵值配對。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[的傳輸閘道是什麼](#)。

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

授與權限給您指定的服務，以代表您在組織內 AWS Organizations 及其帳戶中執行工作。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 AWS Organizations 文件中的[AWS Organizations 與其他 AWS 服務搭配使用](#)。

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

一個小 DevOps 團隊，你可以餵兩個比薩餅。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。

無差別的任務

也稱為繁重工作，是創建和操作應用程序所必需的工作，但不能為最終用戶提供直接價值或提供競爭優勢。無差異化作業的範例包括採購、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

會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。

溫暖的數據

不常存取的資料。查詢此類資料時，通常可以接受中度緩慢的查詢。

視窗功能

一種 SQL 函數，可對以某種方式與當前記錄相關的一組行執行計算。視窗函數對於處理工作非常有用，例如計算移動平均值或根據目前列的相對位置存取列的值。

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 and 應用程式。

蠕蟲

看到[寫一次，多讀](#)。

WQF

請參閱[AWS 工作負載鑑定架構](#)。

寫一次，多讀 (WORM)

一種儲存模型，可單次寫入資料並防止資料遭到刪除或修改。授權用戶可以根據需要多次讀取數據，但無法更改數據。這種數據存儲基礎設施被認為是[不可變的](#)。

Z

零日漏洞

一種利用[零時差漏洞](#)的攻擊，通常是惡意軟件。

零時差漏洞

生產系統中未緩解的瑕疵或弱點。威脅參與者可以利用這種類型的漏洞攻擊系統。由於攻擊，開發人員經常意識到該漏洞。

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。