執行 Amazon Neptune 資料庫ISVs 的多租用戶指引

AWS 方案指引

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 方案指引: 執行 Amazon Neptune 資料庫ISVs 的多租用戶指引

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
資料分割模型 2
Silo-model 3
每個租用戶的叢集 3
孤立模型的實作指引 4
集區模型 6
LPGs的集區模型 7
屬性策略 7
字首標籤策略 9
多標籤策略 11
LPG 模型的效能影響 13
RDF 的集區模型 14
使用圖形存放區 HTTP 通訊協定的 SPARQL 查詢選項 14
RDF 的租用戶隔離 15
準備成長 15
多租用戶案例的限制 16
混合模型 17
最佳實務 18
使用最新版本更新您的 Neptune 叢集 18
使用差異而非刪除和取代資料擷取 18
模擬 Neptune 成本如何隨著租戶而演進 18
擴展叢集以滿足客戶需求 19
後續步驟 20
資源 21
貢獻者 22
文件歷史紀錄 23
..... xxiv

執行 Amazon Neptune 資料庫ISVs 的多租用戶指引

Amazon Web Services（貢獻者)

2024 年 8 月 (文件歷史記錄)

多租戶是一種電腦系統架構，其中應用程式的單一執行個體可服務多個客戶。每個客戶稱為租戶。在多租用戶架構中，這些應用程式執行個體會在共用環境中運作，其中每個租用戶實際上都位於相同的基礎設施上，但在邏輯上是分開的。

身為獨立軟體廠商 (ISV)，您可以使用 Amazon Neptune 來支援需要跨高度連線資料進行導覽的應用程式。您可能正在管理帳戶中的雲端型軟體即服務 (SaaS) 應用程式，並為租戶提供訂閱。租戶接著可以透過網際網路或私下透過 存取服務 AWS PrivateLink。此模型的經濟適用於雙方，因為租戶可以存取成本低於購買、建置和維護成本的軟體。作為 ISV，您可以收取的訂閱費用高於建立和維護軟體的費用。問題是您如何將業務擴展到多個租戶。

多租戶為 ISVs提供重要的經濟和營運優勢。多租戶架構可讓您的組織獲得更好的投資報酬率 (ROI)。多租用戶也簡化了操作需求，讓您的組織可以更快速地移動，並降低將軟體交付給租用戶的成本。

本文件提供使用 Amazon Neptune 有效執行多租戶 ISV 應用程式的指引。本指南是根據多年來，支援 ISVs向客戶成功交付 SaaS 解決方案所取得的最佳實務。在組織目標和架構原則的環境中評估本指南，將協助您找到最佳化解決方案的方法。

> **Note**
>
> 本文件不提供完整的最佳實務清單。它補充了適用於 Amazon Neptune 的 AWS Well-Architected Framework 文件，為多租戶 ISV 工作負載提供額外的特定指導。我們建議您在設計解決方案時檢閱兩份文件中的考量事項。

SaaS 資料分割模型

SaaS 開發人員面臨的挑戰之一，是設計架構模式，以在多租戶環境中代表和組織資料。這些多租戶儲存機制和模式通常稱為資料分割。

在多租戶 SaaS 環境中，區分資料分割和租戶隔離非常重要。這些概念雖然相關，但不是同義字。資料分割是指儲存每個租戶資料的方法。不過，僅分割並不保證租戶隔離。為了確保某個租戶的資料無法供另一個租戶存取，則需要採取其他措施。

多租戶 SaaS 系統中的三種常見資料分割模型是孤立、集區和混合模式。您選擇的任何模型取決於下列因素：

- 合規
- 噪音鄰里
- 分層策略
- 操作需求
- 租用戶隔離需求

此外， 上可用的每個資料庫類型 AWS 通常會提供唯一的資料分割和租用戶隔離模型集合。在查看如何組織租戶圖形以支援解決方案的各種需求時，請考慮 Amazon Neptune 提供的模型。

許多人從下列其中一項聲明ISVs開始在 Neptune 上設計：

- 此ISV解決方案需要跨不同叢集進行客戶實體隔離。
- ISV 解決方案需要建構，例如傳統關聯式資料庫管理系統中找到的具名資料庫或結構描述。

經過考量後，ISVs請了解這些聲明並不正確，因為在幾乎所有工作負載下，每個客戶在資料庫中都有中斷連線的圖表。實作本文件中討論的資料建模和存取指南，可防止跨越這些資料界限並維護客戶資料隱私權。

本指南同時說明孤島模型和集區模型 ，但大多數ISVs人選擇集區模型的成本和操作效率。本指南簡短討論混合模型，結合孤立系統和集區模型的各個層面。有些 為其最大客戶ISVs使用混合模型，以適應圖形大小的法規或合規要求。

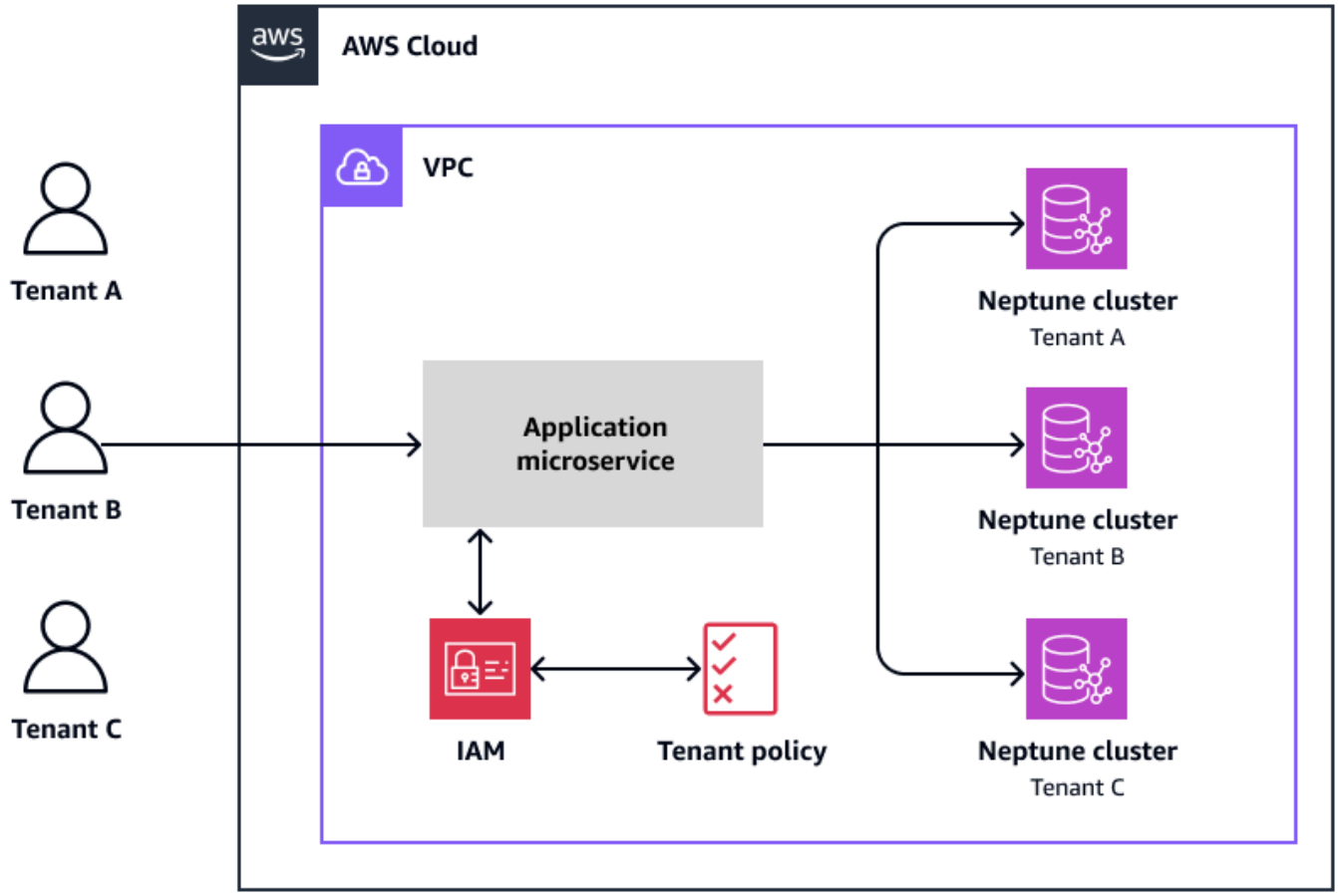
Silo 模型多租用戶

由於合規和法規要求，某些多租戶 SaaS 環境可能需要將租戶的資料部署在完全分開的資源上。在某些情況下，大型客戶需要專用叢集，以減少雜訊鄰近環境的影響。在這些情況下，您可以套用孤立模型。

在孤島模型中，租用戶資料的儲存與任何其他租用戶資料完全隔離。用於代表租用戶資料的所有建構都視為該用戶端的實際唯一，這表示每個租用戶通常都有不同的儲存、監控和管理。每個租用戶也會有用於加密的個別 AWS Key Management Service (AWS KMS) 金鑰。在 Amazon Neptune 中，孤立是每個租用戶一個叢集。

每個租用戶的叢集

您可以使用 Neptune 實作孤立模型，方法是每個叢集有一個租用戶。下圖顯示三個租用戶在虛擬私有雲端 (VPC) 中存取應用程式微服務，每個租用戶各有一個叢集。



每個叢集都有其個別端點，以協助確保不同的存取點，以進行有效率的資料互動和管理。透過將每個租用戶放在自己的叢集中，您可以在租用戶之間建立明確定義的界限，以確保客戶的資料與其他租用戶的

資料成功隔離。此隔離對於具有嚴格法規和安全限制的 SaaS 解決方案也很有吸引力。此外，當每個租用戶都有自己的叢集時，您不必擔心雜訊鄰近，其中一個租用戶施加的負載可能會對其他租用戶的體驗產生負面影響。

雖然cluster-per-tenant孤島模型具有優勢，但它也會帶來管理和敏捷性挑戰。此模型的分散式本質使彙整和評估租戶活動以及所有租戶的操作運作狀態更為困難。部署也會變得更具挑戰性，因為設定新租用戶現在需要佈建個別的叢集。當用戶端升級和版本與資料庫升級緊密結合時，在具有共用用戶端層的環境中升級變得更具挑戰性。

Neptune 同時支援無伺服器和佈建叢集。評估無伺服器或佈建執行個體是否更能處理您的應用程式工作負載。一般而言，如果您的工作負載具有持續的需求層級，則佈建的執行個體將更具成本效益。Serverless 已針對高需求、高度可變的工作負載進行最佳化，其資料庫使用量很短，接著是長時間的輕度活動或沒有活動。

每個租用戶使用 Neptune 佈建叢集時，您必須選取近似租用戶需求最大負載的執行個體大小。這種對伺服器的依賴也會對 SaaS 環境的擴展效率和成本產生層疊影響。雖然 SaaS 的目標是根據實際租用戶負載動態調整大小，但 Neptune 佈建叢集需要您過度佈建，以考慮負載中的大量用量和尖峰期間。過度佈建會增加每個租用戶的成本。此外，隨著租用戶用量隨時間的變化，必須針對每個租用戶分別套用叢集的擴展或縮減。

Neptune 團隊通常會針對孤立模型提供建議，因為閒置資源所產生的成本較高，且具有額外的操作複雜性。不過，對於受到高度管制或敏感的工作負載，客戶可能願意支付額外的成本。

孤立模型的實作指引

若要實作cluster-per-tenant孤立隔離模型，請建立 AWS Identity and Access Management (IAM) 資料存取政策。這些政策透過確保租用戶只能存取包含自己的資料的 Neptune 叢集，來控制對租用戶 Neptune 叢集的存取。將每個租用戶的 IAM 政策連接至 IAM 角色。然後，應用程式微服務會使用 IAM 角色，使用 AWS Security Token Service () `AssumeRole`方法產生精細的臨時登入資料AWS STS。這些登入資料只能存取該租用戶的 Neptune 叢集，用於連線至租用戶的 Neptune 叢集。

下列程式碼片段顯示以資料為基礎的 IAM 政策範例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
```

```
        "neptune-db:WriteDataViaQuery"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:tenant-1-cluster/*",
      "Condition": {
        "ArnEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/tenant-role-1"
        }
      }
    }
  ]
}
```

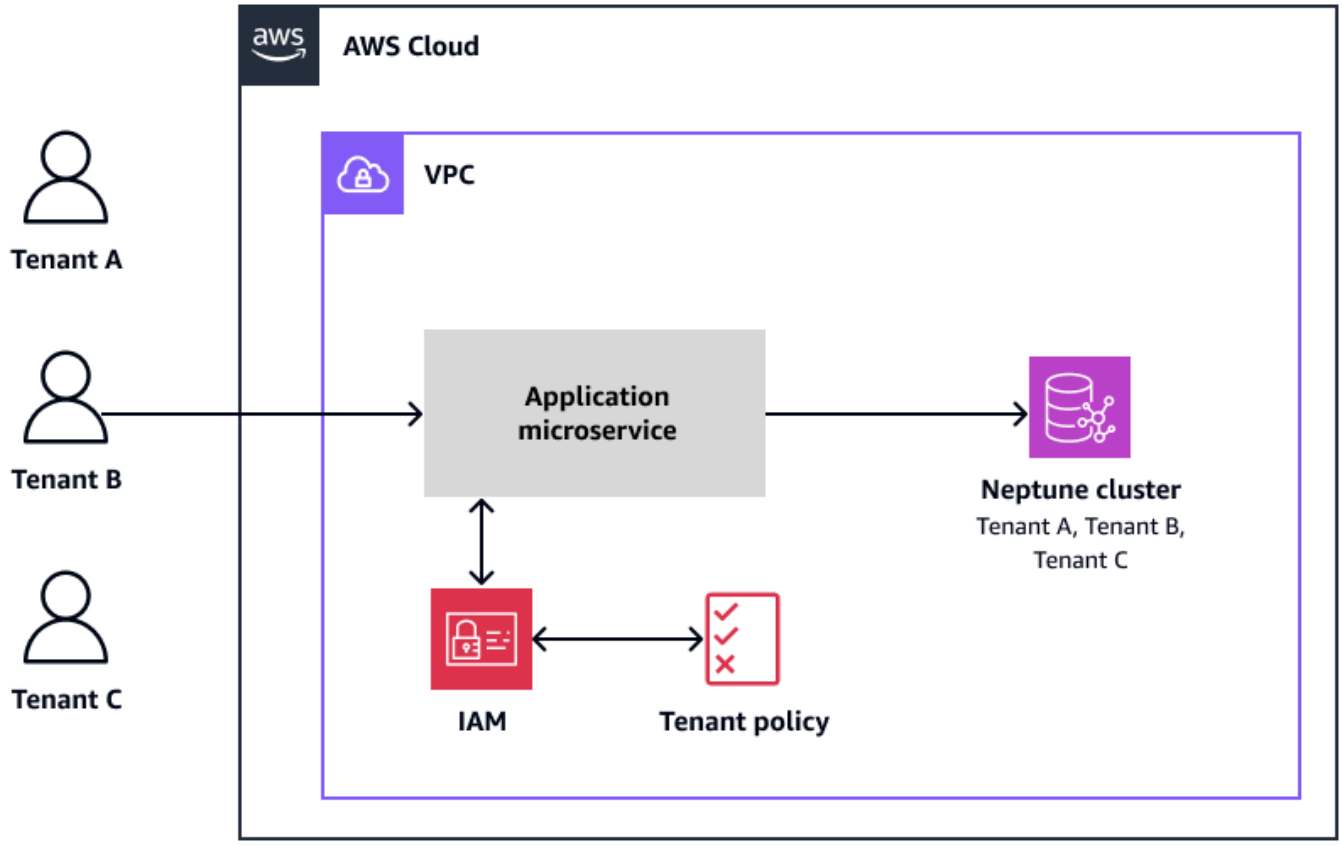
此程式碼提供範例租用戶 `tenant-1`，具有各自 Neptune 叢集的讀取和寫入查詢存取權。`Condition` 元素可確保只有擔任 IAM `tentant-1` 角色 () 的呼叫實體 （委託人`tenant-role-1`) 才能存取 `tenant-1`的 Neptune 叢集。

集區模型多租戶

有時候，由於成本或營運開銷，實作孤立模型並不必要或可行：

- 您可能沒有資源來維護每個租用戶的個別叢集。
- 可能不需要實體分隔每個租戶的資料，而且邏輯分隔足以滿足其需求和合規要求。

下圖顯示集區模型，其中租用戶資料會放置在單一 Amazon Neptune 叢集中，且所有租用戶都會共用共同資料庫。



此集區隔離模型可減少管理開銷，並可提高營運效率，因為管理的叢集較少。此外，運算資源可以跨多個客戶共用，而不是在客戶非作用中期間保持閒置。

當您使用集區模型時，有兩種方式可建立資料模型。您的方法取決於您是使用資源描述架構 (RDF) 建置已標記的屬性圖形 (LPG) 還是圖形。 ???

標記屬性圖表的集區模型

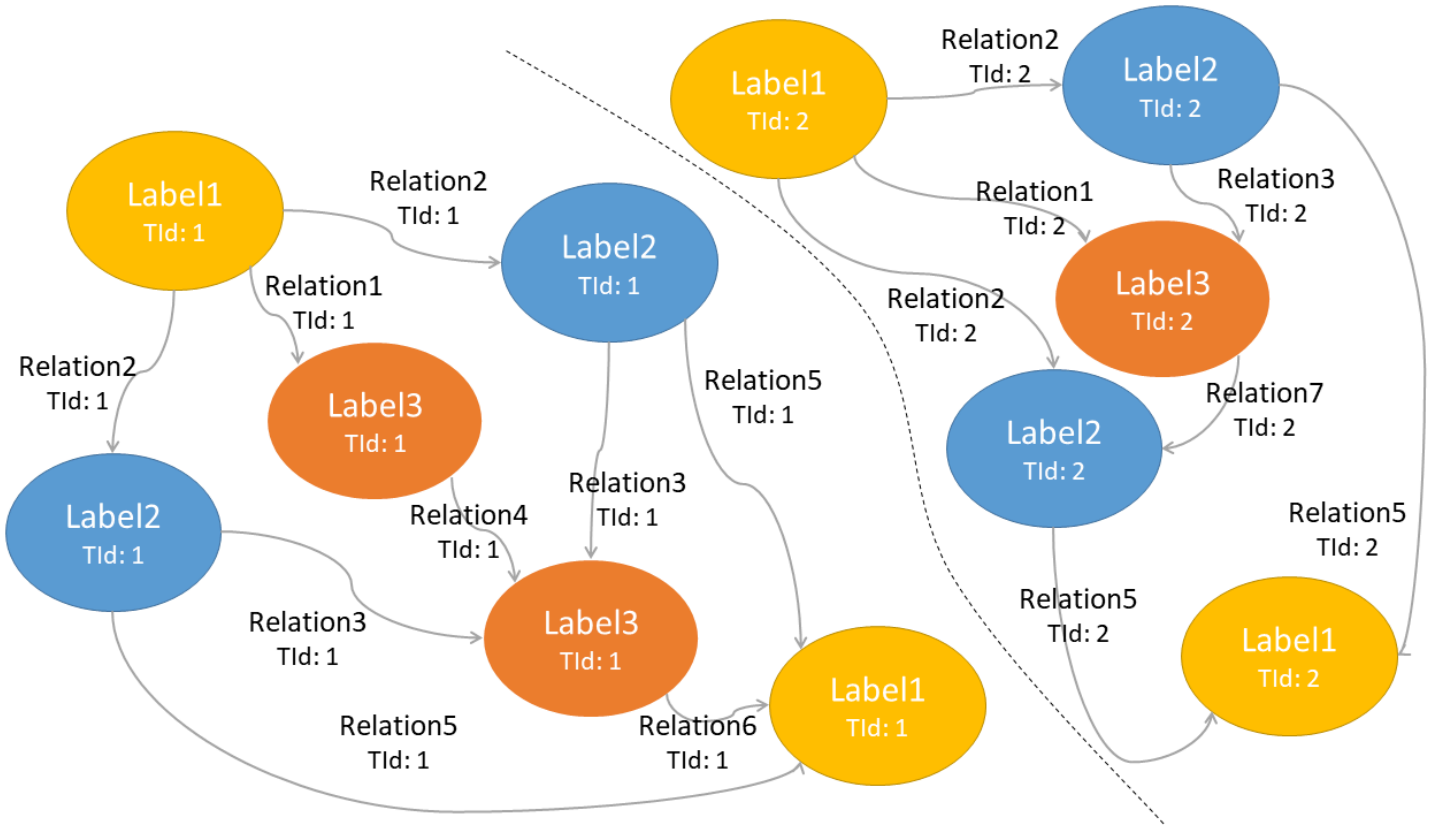
Amazon Neptune 上 LPGs的集區模型有三種不同的方法：

- 屬性策略 – 當您需要優先使用已建立的程式庫建構時，例如 Apache TinkerPop Gremlin 語言的 PartitionStrategy 而非效能，請選擇屬性策略。
- 字首標籤策略 – 根據效能和限制雜訊鄰近效果，我們建議大多數案例使用字首標籤策略。
- 多標籤策略 – 多標籤策略改善了前綴標籤策略的效能。它還支援執行跨叢集上所有租用戶的查詢（例如，用於報告或監控所有租用戶的 ISV 查詢）。

屬性策略

使用 LPGs，使用者可以將鍵值對屬性新增至節點、頂點和邊緣。為了實現邏輯分離，大多數客戶直覺地將此模型化為每個節點和邊緣上具有通用租用戶屬性索引鍵的唯一屬性。租用戶屬性索引鍵代表擁有節點的所有租用戶。租用戶識別符是識別個別租用戶的唯一值。

下圖顯示此模型。兩個中斷連接的子圖具有各種標記的節點和邊緣，租用戶屬性索引鍵由 表示`TId`。一個子圖的每個節點和邊緣都有 `TId`的值`1`。在另一個子圖中，每個節點和邊緣`TId`的值為 2。



在標記的屬性圖表中，有兩種管理此項目的方法。Gremlin 查詢語言提供 PartitionStrategy 周遊程式庫，以協助管理資料的資料分割。下列範例中的程式碼預期每個節點和邊緣都有名為 的屬性TId：

```
strategy1 = new PartitionStrategy(partitionKey: "TId", writePartition: "1",
 readPartitions: ["1"])
strategy2 = new PartitionStrategy(partitionKey: "TId", writePartition: "2",
 readPartitions: ["2"])
```

寫入新節點或邊緣時，會根據是否strategy1strategy2選取 "1"或 "2"，以 或 的值"TId"新增 屬性。對於使用 "TId"的客戶"1"，您可以使用 strategy1。下列範例顯示為該客戶寫入資料：

```
g.withStrategies(strategy1).addV("Label1").property("Value", "123456").property(id,
 "Item_1")
```

對於讀取查詢， "TId == '1'"或 的篩選條件"TId == '2'"會strategy2分別使用 strategy1或 新增至每個節點或邊緣周遊。這些分割區策略可簡化您的程式碼，但並非必要。使用策略的好處是它可以在授權層級注入，並傳遞至形成查詢的較低層級程式碼。這會將決定客戶識別符 (TId) 的程式碼與查詢的邏輯分開。

下列範例程式碼顯示讀取資料的 Gremlin 查詢：

```
g.withStrategies(strategy1).V().hasLabel("Label1")
```

上述程式碼等同於下列範例：

```
g.V().hasLabel("Label1").has("TId", "1")
```

同樣地，使用 Gremlin 寫入資料時，您可以使用下列查詢：

```
g.withStrategies(strategy1).addV("Label1").property("Value").property(id, "Item_1")
```

上述程式碼等同於下列範例，該範例不使用 分割區策略，因此需要明確寫入 "TId" 屬性：

```
g.addV("Label1").property("TId", "1").property("Value").property(id, "Item_1")
```

在 openCypher 中，這些程式庫不存在。您有責任撰寫和修改查詢，將租戶識別符新增為節點和邊緣的屬性。例如：

```
CREATE (n:Item {`~id`: 'Item_1', Value: '123456', TId: '1'})
CREATE (n:Item {`~id`: 'Item_2', Value: '123456', TId: '2'})
```

請注意 Gremlin 程式碼之間沒有分割區策略的相似性。然後，您可以使用下列程式碼讀取從第一個CREATE陳述式寫入的節點：

```
MATCH (n:Item {TId: '1'})
RETURN n
--or
MATCH (n:Item)
WHERE n.TId == '1'
RETURN n
```

當您想要使用原生 TinkerPop Gremlin 建構，例如 PartitionStrategy 時，可以選擇 屬性策略。不過，相較於前綴標籤策略，此模型在 Amazon Neptune 上有效能缺點。如需這些效能缺點的討論，請參閱 LPG 模型的效能影響一節。

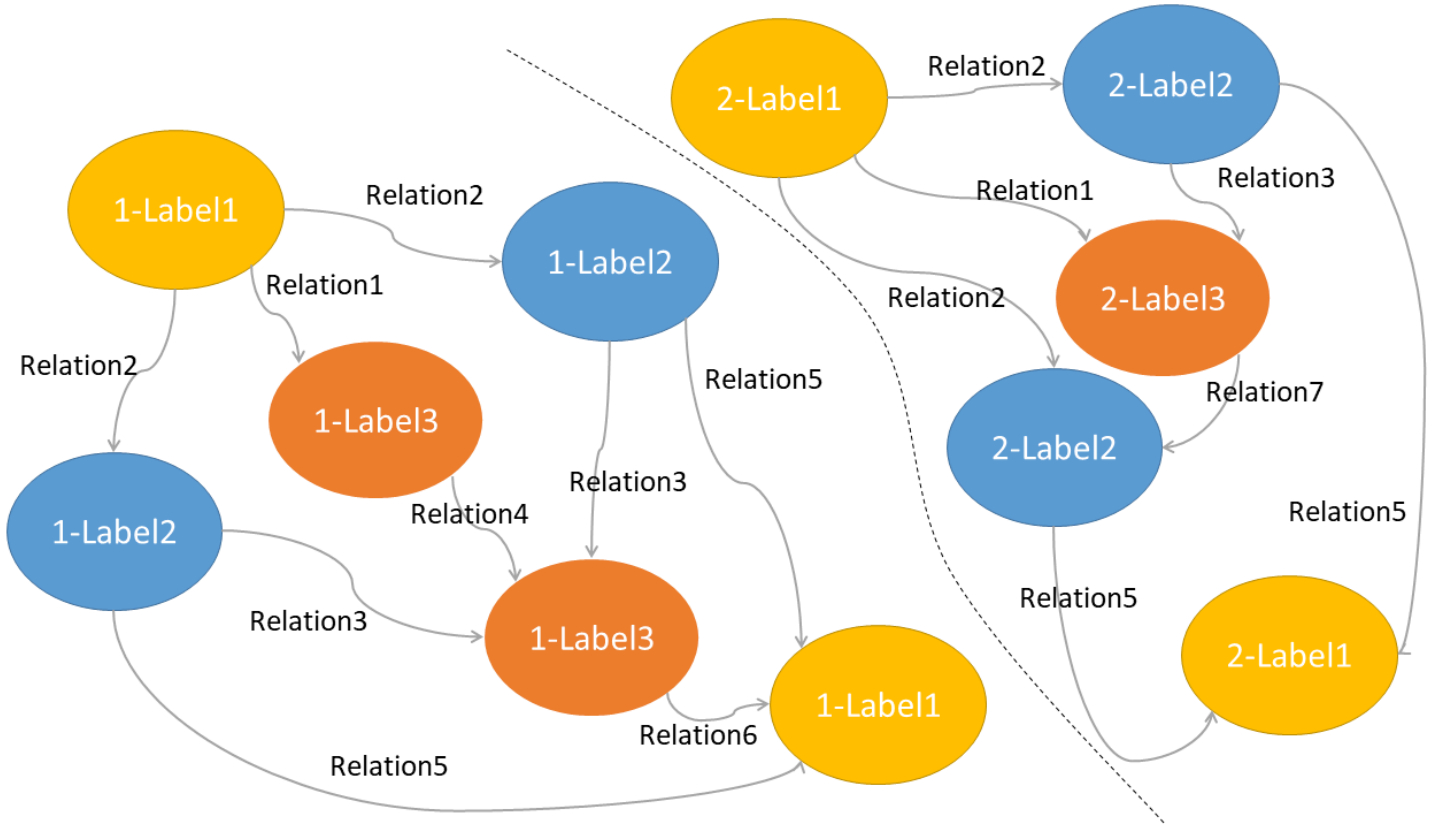
如果符合下列條件，請考慮僅在節點上建模屬性策略，而不是在邊緣上建模：

- 您的圖形邊緣明顯比標籤多。
- 每個租用戶都是中斷連線的圖形。
- 您只能使用節點做為起點，而不是標籤來存取圖形。

字首標籤策略

如果效能是首要考量，強烈建議您考慮屬性策略的前綴標籤策略。

在字首標籤策略中，您可以使用租戶識別符和節點標籤的組合來標記每個節點。例如，如果租用戶的識別符為 "1"且節點標籤為 "Label1"，您可以將節點標籤指定為 "1-Label1"。下圖顯示兩個使用此模型的中斷連線子圖。



在 Gremlin 中寫入資料時，您可以將識別號碼新增至任何節點的標籤：

```
g.addV("1-Label1")
g.addV("2-Label6")
```

查詢此圖形時，您可以檢查節點上是否存在此字首：

```
g.V().hasLabel("1-Label1")
```

在 openCypher 中，您可以使用CREATE陳述式寫入資料：

```
CREATE (n:`1-Label1` {`~id`: 'Item_1', Value: 'XYZ123456'})
```

若要查詢您在 openCypher 中撰寫的資料，請使用下列程式碼：

```
MATCH n= (:`1-Label1`)
RETURN n
```

字首標籤策略假設所有節點都指派給一或多個租用戶，並且未在邊緣範圍內指派許可。避免在邊緣標籤上使用此策略，因為這會導致大量的述詞，並對 Neptune 效能產生負面影響。

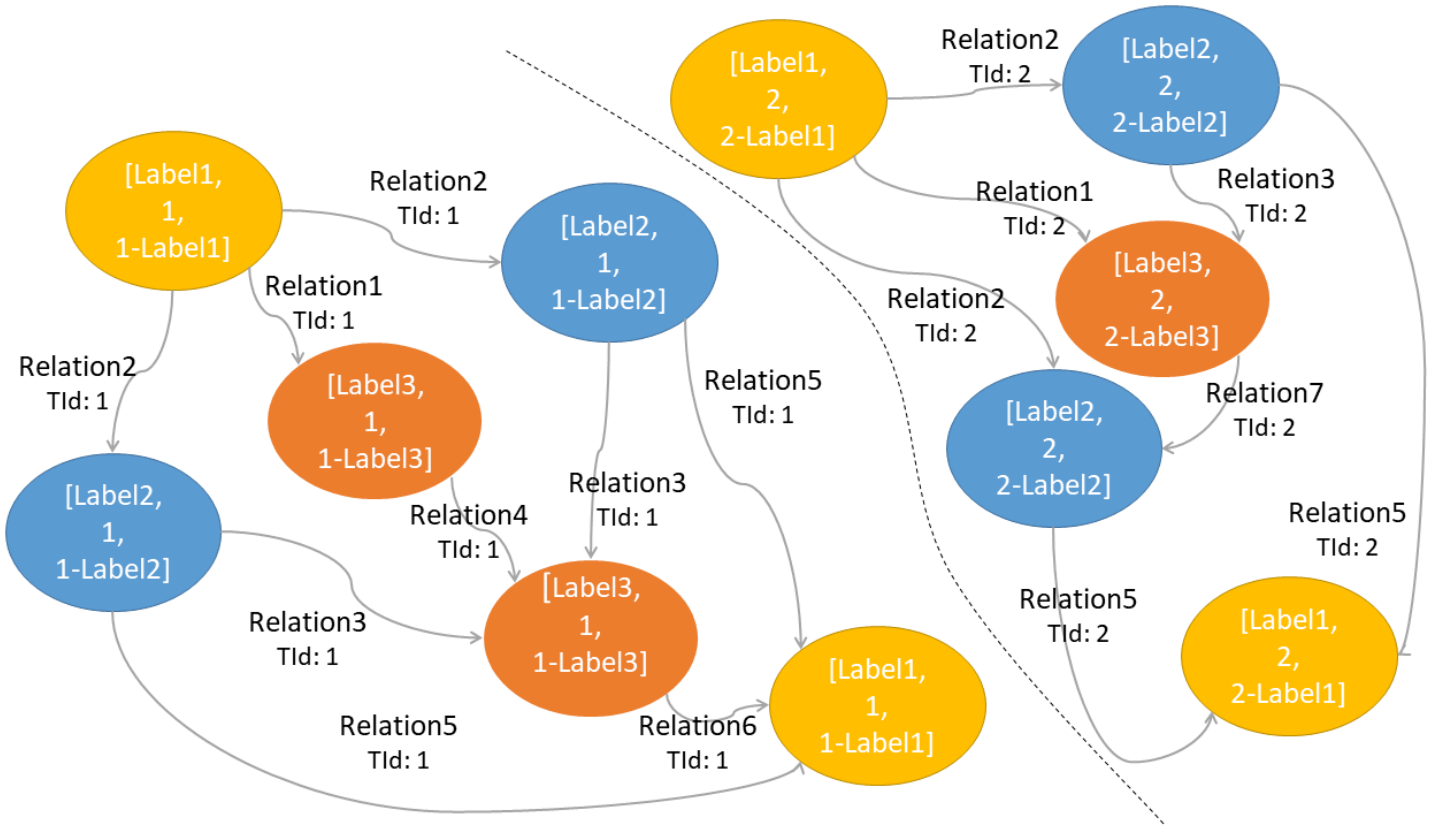
字首標籤方法有兩個主要缺點。首先，執行跨租用戶的任何查詢並不容易。例如，查詢會計算指定標籤的所有節點以進行報告或監控。如果這是您的使用案例，請考慮將此策略與多標籤策略結合。如需結合策略的詳細資訊，請參閱混合模型一節。

其次，字首標籤策略需要控制，強制對每個查詢適當套用適當的字首，以防止資料外洩。不過，對於需要低延遲查詢的工作負載而言，此策略是最有效率的選項，我們強烈建議這麼做。LPG 模型的效能影響區段提供為什麼這是最有效策略的範例。

多標籤策略

第三個選項是使用多標籤策略。對於此方法，您可以將額外的標籤新增至圖形上的每個節點。例如，如果您需要篩選指定租用戶的所有資料，請新增租用戶 ID 標籤。 如果您需要篩選指定標籤的所有資料，無論租用戶為何，請新增該標籤。下圖顯示針對每個節點使用三個標籤所套用的多標籤策略。

您現在可以使用三種不同的模式來存取圖形：



- 篩選 `Label1`以傳回所有租用戶`Label1`中具有 的所有節點。
- 篩選 `1`以傳回租戶 1 的所有節點。
- 篩選 `1-Label1` 以傳回僅具有標籤 之租用戶 1 的所有節點`Label1`。

對於 LPGs，有兩種方法可以實作此項目。

在 Gremlin 中，您可以使用名為 SubgraphStrategy 的周遊策略，將所有查詢的範圍限制為僅具有特定標籤的頂點，例如 `"Label1"`：

```
g.withStrategies(
    new SubgraphStrategy(
        vertices=hasLabel("Label1")
    )
)
```

與 PartitionStrategy 不同，SubgraphStrategy 只會影響讀取資料，不會影響寫入資料。若要寫入資料，請在每個查詢中手動指派標籤：

```
g.addV("Label1").property("Value","XYZ123456")
.addV("Label2").property("Value","XYZ123456")
```

讀取資料時，您可以使用 SubgraphStrategy 透過 查詢所有節點`"Label1"`：

```
g.withStrategies(
new SubgraphStrategy(vertices=.hasLabel("Label1"))
).
V().has("Value","XYZ123456")
```

Neptune 只會傳回第一個記錄，其具有 `"Label1"`和 的值`"XYZ123456"`。它相當於下列查詢，不使用 SubgraphStrategy：

```
g.V().hasLabel("Label1").hasValue("XYZ123456")
```

在此基本查詢中，SubgraphStrategy 使用起來更為複雜。請記住，您的程式庫可以提供已定義策略g的執行個體。開發人員不必確保套用適當的篩選條件：

```
def getGraphTraversal():
return g.withStrategies(new SubgraphStrategy(vertices=.hasLabel("Label1"))

getGraphTraversal().has("Value","XYZ123456")
```

openCypher 程式庫沒有這些建構，因此您必須為每個節點建立多個標籤：

```
CREATE (n:`1`:`Label1`:`1-Label1` {`~id`: 'Item_1', Value: '12345'})
```

當您使用這些標籤篩選子圖時，您可以傳回具有您要尋找之客戶標籤的節點，或與具有該標籤的其他節點共用關係的節點：

```
MATCH n=(:Label1:`1`)
// or
MATCH n=(:`1-Label1`)
```

多標籤策略可讓您根據類型 (`Label1`) 或租用戶 () 來查詢節點，或在效能最為重要 (1) 時使用更有效率的字首標籤策略`1-Label1`。

此策略的主要缺點是每個標籤都是存放在圖形中的額外物件。物件是 LPGs 中節點或邊緣上的節點、邊緣或屬性。擷取速度由每秒物件測量和限制，儲存成本取決於消耗的 GB 數。這表示額外的物件可能會產生大規模的可測量影響。

LPG 模型的效能影響

Amazon Neptune 的 AWS 技能建置器課程資料建模詳細說明了 Neptune 資料模型內部和建模影響，但我們將在此摘要說明這些設計的重要考量事項。 Amazon Neptune 考慮在單一 Neptune 叢集上擁有三個租用戶 (T1、T2、T3)。這些租用戶具有下列屬性：

- 租用戶 1 (T1) 總共有 1 億個節點，1000 萬個是類型項目。
- 租用戶 2 (T2) 總共有 1，000 萬個節點，100 萬個是類型項目。
- 租戶 3 (T3) 總共有 1 億個節點，100 萬個是類型項目。

執行查詢，該查詢將使用 屬性策略擷取租用戶 3 的項目。Neptune 會檢查兩個索引呼叫的統計資料：

- 其中 `tenant property key=T3`有 1 億個結果
- 其中 `label = Item`有 1，200 萬筆結果 (1，000 萬筆來自 T1 + 100 萬筆來自 T2 + 100 萬筆來自 T3)

Neptune 查詢最佳化工具會判斷後者查詢最適合先套用 (1，200 萬個結果），然後檢查每個項目是否有 `tenant property key=T3`。您可以擷取 1，200 萬個項目來尋找 100 萬個結果。

請注意此查詢的雜訊相鄰影響。如果您每個租用戶有 1 億個項目節點，第一個查詢會有 3 億個結果，而不是 1，200 萬個 （為了說明目的，這過度簡化了。 Neptune 最佳化工具可能已套用不同的操作順序）。

接著，請考慮字首標籤策略。進行單一索引呼叫`label=T3-Item`，其中 會 傳回 100 萬個結果。這可實現與屬性策略相同的結果，但擷取的記錄減少 1，100 萬筆。此外，您不再有雜訊的鄰里問題，因為標籤在索引中不會重疊。

多標籤策略不會直接改善屬性策略的查詢效能。當搜尋空間也相當時，依屬性值篩選與依標籤值篩選相當。反之，多標籤策略支援更多彈性。 多標籤策略提供的效能等同於 `label=T3`或標籤 的前綴標籤策略`T3-Item`。多標籤策略提供的效能等同於 的屬性策略`label=Item`。好處是支援各種存取模式。

RDF 的集區模型

資源描述架構 (RDF) 具有具名圖形的概念，可提供分隔資料的邏輯方式。在 Amazon Neptune 中，您有預設的命名圖形和使用者定義的命名圖形。您可以視需要建立任意數量的具名圖形。統稱為 RDF 資料集。預設或使用者定義的所有具名圖形都是由 RDF 資料集內的國際化資源識別符 (IRI) 定義。在 Neptune 中，除非使用者在寫入資料時宣告具名圖形，否則所有三元組都會被視為預設具名圖形的一部分。

具名圖形有多個使用案例：

- 資料分割和資料隔離
- 資料來源
- 版本控制
- Inference

本指南著重於資料分割使用案例。我們建議為每個租用戶建立一個使用者定義的具名圖形。

使用圖形存放區 HTTP 通訊協定的 SPARQL 查詢選項

下列範例查詢使用 SPARQL 通訊協定和 RDF 查詢語言 (SPARQL) 和圖形存放區 HTTP 通訊協定來查詢或建立租用戶的具名圖形。

- `HTTP GET` – 若要擷取租用戶的特定圖形：

```
curl --request GET 'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//
www.example.com/named/tenant1'
```

- `HTTP PUT` – 若要使用請求中指定的承載建立或取代特定具名圖形：

```
curl --request PUT -H "Content-Type: text/turtle" \  --data-raw "@prefix ex: http://
example.com/ . ex:subject ex:predicate ex:object ." \
```

```
'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/
tenant1'
```

在 RDF 中，物件是三元物件。

- HTTP POST – 若要在圖形不存在時建立新的具名圖形，或與現有圖形合併：

```
curl --request POST -H "Content-Type: text/turtle" \
--data-raw "@prefix ex: http://example.com/ . ex:subject ex:predicate ex:object ." \

'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/
tenant1'
```

RDF 的租用戶隔離

若要在應用程式層使用必要的護欄對資料進行邏輯隔離，請在租用戶和使用者定義的具名圖形之間建立映射。當您為 RDF 資料集設計多租戶時，請注意 RDF 和 SPARQL 的下列層面：

- 在 Neptune 中，當您在不指定具名圖形的情況下進行查詢時，它會擷取資料庫中所有具名圖形中符合模式的所有三元組。
- 在 RDF 中，不同具名圖形節點之間的連線沒有限制。例如，在上圖中， 中的節點`:G1`可以透過邊緣連接到 `G2`中的節點。

例如，如果特定租用戶的最終使用者提交查詢至 API，API 應先驗證下列需求，再將查詢提交至 Neptune 資料庫：

- 任何範圍在單一租用戶的查詢都必須指定具名圖形。否則，您會面臨跨租用戶洩漏資料的風險。
- 更新或刪除查詢應一律指定具名圖形。
- 邊緣或關係任一端的節點應一律屬於正確的具名圖形。

如需最佳實務的詳細資訊，請參閱 Neptune 文件。

準備成長

當您成功使用集區模型時，您最終會超出單一 Neptune 叢集的大小。租用戶增加，或租用戶數量增加，而且所有客戶所需的資料擷取速率都超過叢集的功能。發生這種情況時，您需要將客戶分割到多個

叢集。預先為此組態進行設計，而不是稍後嘗試進行修改。即使您的初始擴展是僅使用單一叢集， 仍會模擬您未來達到該擴展時，將租用戶路由到多個叢集所需的元件。

如果您的解決方案根據您的租戶大小需要更多資源，也請為他們的成長做好準備。如果單一叢集上的數個客戶大幅成長，該叢集可能不再支援您的需求。設計策略，使用 Amazon Neptune 資料庫複製功能將租戶移至另一個叢集，或將現有叢集分割成兩個叢集。

熟悉 Neptune Copy-on-Write協定，這可在您實作資料庫複製時節省成本。如果您因為擷取瓶頸而分割叢集，則不從叢集刪除資料可能會更有效率，前提是您的政策允許這樣做。如果資料頁面保持不變，但如果資料頁面遭到修改（因為其中的某些資料遭到刪除），這兩個叢集將會共用資料頁面。

> **Note**
>
> 本指南適用於撰寫本文時的最新 Neptune 版本，即 Neptune 1.3.1 版。隨著 Neptune 儲存層的演進，本指南可能會在未來版本中變更。

多租用戶案例的限制

請注意，某些 Neptune 功能並非針對多租用戶案例而建置。租用戶不應直接存取集區模型中的 Neptune 端點，因為這些多租用戶策略未在資料庫層級強制執行。一律在客戶與 Neptune 端點之間保留某種代理，以強制執行本文件中所述的設計。這類代理的範例包括下列項目：

- 在用戶端層中附加標籤篩選條件
- 擁有將身分驗證字符映射至租戶 ID 並將此篩選條件插入查詢的 API

本指南也適用於讓客戶直接存取 Neptune 圖形筆記本、Neptune 圖形探索程式或 Neptune 串流等功能。

混合模型多租戶

SaaS 解決方案通常會使用孤島和集區模型的混合。各種因素會影響何時以及如何在相同環境中使用孤立系統和集區模型的決定。

其中一個因素是分層，其中 SaaS 解決方案為每個租戶層提供獨特的體驗。例如，如果您的層是免費、標準和高級，您的免費層租用戶資料可能會使用集區模型儲存在共用的 Neptune 叢集中。對於 Standard 和 Premium 層租用戶，您可以使用 cluster-per-tenant 孤島模型。

此外，某些 SaaS 供應商也可以在共用的 Amazon Neptune 叢集上建置其集區解決方案，作為其基礎。隨後，他們可以為需要孤立儲存的租戶建立單獨的 Neptune 叢集，通常是因為合規和監管要求。

雖然這可能會為您的資料存取層和管理設定檔增加複雜程度，但它也可以為您的企業提供一種方法來將產品分層以滿足客戶要求。

ISVs的操作最佳實務

本節中的許多指導方針適用於所有客戶的最佳實務，但它們已為 ISVs增加重要性。

使用最新版本更新您的 Neptune 叢集

在 Amazon Neptune 版本備註中，您可以看到每個版本都具有許多錯誤修正、效能改善和新功能。將 Neptune 叢集盡可能保持在最新版本上。

如果您在工作負載中發現先前未發現的錯誤，且叢集位於最新版本上，Neptune 工程師可以為您的叢集建立私有修補程式（如果需要，且您想要）。修補程式可以橋接，直到該修正正式推出下一個版本為止。為了協助您將叢集更新至最新版本，請使用 Neptune 藍/綠解決方案。

使用差異而非刪除和取代資料擷取

您可以使用數種技術，將資料擷取或寫入 Neptune。許多客戶嘗試在每次收到變更時刪除並重新插入圖形，以簡化其資料擷取。它們可能會將`last-modified`屬性新增至每個節點，並定期掃描自某些指定日期以來尚未修改的節點，然後刪除它們。雖然這些技術簡化了資料擷取程序，但它們對您的 Neptune 叢集具有長期運作狀態和可擴展性影響。

首先，Neptune 使用字串的字典編碼。除非您明確指定節點和邊緣IDs，否則 Neptune 會產生以 ID 字串表示的 GUID，並將該字串存放在字典中。如果您持續刪除和新增物件，自動產生的 IDs 會在字典中造成膨脹。

其次，Neptune 可擴展至每秒最大擷取約 120 K 個物件。如果您持續刪除並新增物件，則會在實質上不會變更的物件上消耗大量頻寬。這會限制您可以在叢集上託管的租用戶數量、需要叢集中較大的寫入器執行個體，以及需要更多 I/O 操作。所有這些因素都會增加您的成本。

強烈建議您開發一種方法來計算已變更的真實差異，而不是使用刪除和新增方法。不過，某些資料來源不利於此（例如，傳回目前狀態的 API 呼叫，或是未確切追蹤變更的事件）。如果您的原始資料來源不有利於識別變更，請使用擷取、轉換和載入 (ETL) 程序來計算差異。例如，您可以維持 Parquet 格式每個先前資料擷取的快照，使用 AWS Glue 計算這些快照之間的差異，並僅將差異推送至 Neptune。

模擬 Neptune 成本如何隨著租戶而演進

無論您使用孤立、集區或混合模型，雲端成本都會隨著租戶的大小而擴展。需要更多並行連線的租用戶需要比較少並行連線的執行個體或多個僅供讀取複本。這同樣適用於需要更快速資料擷取的租戶。

Neptune 叢集成本的三個元件包括執行個體大小（和數量）、資料大小 (GB 月）和 I/O 操作（每百萬）。雖然這些成本通常是工作負載特定的，但它們會隨著大小和資料量而擴展，但可以使用 AWS 工具來測量。根據租戶大小的關鍵指標來追蹤和了解規模經濟，包括其大小隨著時間的變化。如果 I/O 費用的不可預測性影響您的利潤，請考慮選擇 Neptune I/O 最佳化儲存，以獲得更可預測的成本。

擴展叢集以滿足客戶需求

正確調整 Neptune 執行個體大小沒有嘗試過或真正的公式。Neptune 文件提供指引，但有太多變數可以建議直接映射。這些變數包括但不限於下列項目：

- 資料模型
- 資料形狀
- 查詢並行
- 查詢複雜性。

規劃測試，以判斷工作負載和租戶設定檔的最佳大小。一般而言，我們建議使用佈建的執行個體，以實現成本效益和可預測性。如果您的客戶體驗目標優先於成本的最佳擴展，請考慮使用 Neptune Serverless 執行個體，以確保無論工作負載波動如何，都能獲得更一致的體驗。

如果您的租戶讀取工作負載的尖峰和低谷有顯著差異，請將 Neptune Serverless 執行個體與 Neptune 自動調整規模結合。 新的僅供讀取複本初始化後，通常需要 10-15 分鐘才能上線。這表示僅自動擴展可以處理流量長時間變更，但不足以快速變更活動的峰值。透過結合 Neptune Serverless 和 Neptune 自動擴展，您可以向上或向下擴展執行個體，並向外擴展僅供讀取複本的數量。

如果您的租戶有顯著不同的工作負載描述檔或服務水準協議 (SLAs)，請考慮使用自訂端點和專用僅供讀取複本，將流量導向至該流量最佳化的執行個體。最佳化可包含不同大小的執行個體、特定查詢模式，或預暖緩衝快取。

後續步驟

如果您剛開始為多租用戶ISV應用程式實作 Amazon Neptune 的旅程，請將額外的考量納入您想要的模型。變更模型之後在旅程中會更昂貴。

如果您在旅程中提早，請確認您使用符合您需求的最佳模型，以及遵循該模型的指引。

請事先計畫。當您在旅程的早期時，很誘人地會延遲跨叢集共用客戶或最佳化ETL程序的工作，以提供變更的差異，而不是刪除和重新新增頂點和邊緣。當您擴展時，這些決策可能會對效能和成本產生負面影響。

最後，如果您已順利完成旅程，本指南可能會向您保證您的架構是最佳的，或者它可能會提供變更來改善您的架構。

如果您對本指南有疑問或需要進一步協助，請聯絡您的 AWS 帳戶 團隊，並要求與 Neptune 專家進行工作階段。

資源

- Amazon Neptune 文件
- Amazon Neptune 的資料建模（課程）
- 套用 AWS Well-Architected Framework for Amazon Neptune
- SaaS Lens Well-Architected 架構
- 上的多租戶架構指南 AWS
- SaaS 租戶隔離策略：在多租戶環境中隔離資源
- Apache TinkerPop 文件
- SPARQL

貢獻者

本指南的貢獻者包括：

- Brian O'Keefe，校長 WW SSA Neptune， AWS
- Veeresham Gande，資深技術客戶經理， AWS
- Dana Owens，啟動解決方案架構師， AWS
- Nima Seifi，啟動解決方案架構師， 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 9 月 3 日

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。