使用 **REST** 實現自動化

ONTAP Select

NetApp
May 07, 2026

目錄

使用 **REST** 實現自動化

概念

用於部署和管理 **ONTAP Select** 叢集的 **REST Web** 服務基礎

Representational State Transfer（REST）是一種用於建立分散式 Web 應用程式的風格。當應用於 Web 服務 API 的設計時、它建立了一套技術和最佳實務做法、用於公開伺服器型資源並管理其狀態。它使用主流傳輸協定和標準、為部署和管理 ONTAP Select 叢集提供靈活的基礎。

架構與傳統限制

REST 由 Roy Fielding 在 2000 年於 UC Irvine 攻讀博士學位 "論文"期間正式提出。它透過一系列約束定義了一種架構風格，這些約束共同改進了基於 Web 的應用程式及其底層協定。這些約束建立了一種基於客戶端/伺服器架構並使用無狀態通訊協定的 RESTful Web 服務應用程式。

資源和狀態表示法

資源是網路型系統的基本元件。建立 REST Web 服務應用程式時，早期設計工作包括：

- 系統或伺服器資源的識別 每個系統都會使用和維護資源。資源可以是檔案、業務交易、流程或管理實體。基於 REST Web 服務設計應用程式的首要任務之一就是識別資源。
- 資源狀態及其相關狀態操作的定義 資源始終處於有限數量的狀態之一。必須明確定義這些狀態以及用於影響狀態變化的相應操作。

用戶端和伺服器之間交換訊息,以根據通用的 CRUD (Create、Read、Update 和 Delete) 模型存取和變更資源的狀態。

URI 端點

每個 REST 資源都必須使用定義完善的定址方案進行定義和提供。資源所在和識別的端點使用統一資源識別碼（URI）。URI 提供了一個通用框架，用於為網路中的每個資源建立唯一名稱。統一資源定位器（URL）是一種與 Web 服務搭配使用的 URI 類型，用於識別和存取資源。資源通常以類似於檔案目錄的階層式結構公開。

HTTP 訊息

超文本傳輸協定（HTTP）是 Web 服務用戶端和伺服器用於交換資源請求和回應訊息的協定。在設計 Web 服務應用程式時，HTTP 動詞（例如 GET 和 POST）會被對應到資源和對應的狀態管理操作。

HTTP 是無狀態的。因此，要將一組相關的請求和回應關聯到一個交易中，必須在請求/回應資料流攜帶的 HTTP 標頭中包含額外資訊。

JSON 格式化

雖然資訊可以透過多種方式在客戶端和伺服器之間進行結構化和傳輸，但最常用的方式（也是 Deploy REST API 使用的方式）是 JavaScript 物件表示法（JSON）。JSON 是一種行業標準，用於以純文字形式表示簡單的資料結構，並用於傳輸描述資源的狀態資訊。

如何存取 **ONTAP Select Deploy API**

由於 REST Web 服務固有的靈活性，可以透過多種不同的方式存取 ONTAP Select Deploy API。

ONTAP Select Deploy 中所包含的 REST API 被指派了一個版本號碼。此 API 版本號與 Deploy 版本號無關。ONTAP Select 9.17.1 Deploy 管理公用程式包含 REST API 的版本 3。

部署公用程式原生使用者介面

存取 API 的主要方式是透過 ONTAP Select Deploy Web 使用者介面。瀏覽器會呼叫 API 並根據使用者介面的設計重新格式化資料。您也可以透過 Deploy 公用程式命令列介面存取 API。

ONTAP Select Deploy 線上文件頁面

使用瀏覽器時，ONTAP Select Deploy 線上文件頁面提供了另一種存取途徑。除了可以直接執行各個 API 呼叫之外，頁面還包含 API 的詳細描述，包括每個呼叫的輸入參數和其他選項。API 呼叫以不同的功能領域或類別進行組織。

自訂程式

您可以使用多種程式語言和工具存取 Deploy API。常用的語言包括 Python、Java 和 cURL。使用該 API 的程式、腳本或工具可作為 REST Web 服務用戶端。使用程式語言可以幫助您更好地理解 API，並有機會實現 ONTAP Select 部署的自動化。

ONTAP Select Deploy API 基本操作特性

雖然 REST 建立了一套通用的技術和最佳實務做法，但每個 API 的具體細節會因設計選擇而異。在使用 ONTAP Select Deploy API 之前，您應該了解其詳細資訊和操作特性。

Hypervisor 主機與 **ONTAP Select** 節點

hypervisor host 是承載 ONTAP Select 虛擬機器的核心硬體平台。當 ONTAP Select 虛擬機器部署並在 hypervisor host 上作用中時，此虛擬機器被視為 *ONTAP Select* 節點。在 Deploy REST API 版本 3 中，主機和節點物件是分離且不同的。這允許一對多的關係，即一個或多個 ONTAP Select 節點可以在同一個 hypervisor host 上執行。

物件識別碼

每個資源執行個體或物件在建立時都會被指派一個唯一識別碼。這些識別碼在 ONTAP Select Deploy 的特定執行個體中是全域唯一的。發出建立新物件執行個體的 API 呼叫後,相關的 ID 值會在 HTTP 回應的 `location` 標頭中傳回給呼叫方。您可以擷取該識別碼,並在後續呼叫中參照該資源執行個體時使用它。

物件識別碼的內容和內部結構可能隨時變更。在引用關聯物件時，您應該僅在需要時在對應的 API 呼叫中使用這些識別碼。

請求識別碼

每個成功的 API 請求都會被指派一個唯一的識別碼。該識別碼會在相關聯的 HTTP 回應的 `request-id`標頭中傳回。您可以使用請求識別碼來統稱單一特定 API 請求-回應交易的活動。例如、您可以根據請求 ID 擷取交易的所

有事件訊息。

同步和非同步呼叫

伺服器處理從客戶端收到的 HTTP 請求主要有兩種方式：

- 同步伺服器會立即執行要求，並以狀態碼 200、201 或 204 回應。
- 非同步處理：伺服器接受請求並傳回狀態碼 202。這表示伺服器已接受客戶端請求，並啟動後台任務以完成該請求。最終的成功或失敗結果不會立即顯示，必須透過額外的 API 呼叫來確定。

確認長時間執行工作的完成

通常，任何耗時較長的操作都會在伺服器端使用後台任務非同步處理。在 Deploy REST API 中，每個後台任務都由一個 Job 物件關聯，該物件追蹤任務並提供諸如當前狀態之類的資訊。後台任務建立後，HTTP 回應會傳回一個包含其唯一識別碼的 Job 物件。

您可以直接查詢 Job 物件，以判斷相關 API 呼叫是否成功。如需其他資訊，請參閱_使用 Job 物件進行非同步處理_。

除了使用 Job 物件之外，還有其他方法可以判斷要求是否成功或失敗，包括：

- 事件訊息您可以使用原始回應中傳回的請求 ID 來擷取與特定 API 呼叫關聯的所有事件訊息。事件訊息通常包含成功或失敗的指示，在偵錯錯誤情況時也很有用。
- 資源狀態或狀態 許多資源都維護一個狀態或狀態值、您可以查詢該值以間接確定請求是否成功或失敗。

安全性

Deploy API 使用以下安全技術：

- 傳輸層安全性協定 (TLS) Deploy 伺服器和用戶端之間透過網路傳輸的所有流量均透過 TLS 加密。不支援透過未加密通道使用 HTTP 協定。支援 TLS 1.2 版。
- HTTP 驗證基本驗證用於每個 API 交易。每個請求都會新增一個 HTTP 標頭，其中包含 base64 字串中的使用者名稱和密碼。

ONTAP Select 的請求和回應 API 交易

每次 Deploy API 呼叫都是向 Deploy 虛擬機器發送 HTTP 請求，虛擬機器隨後會產生相應的回應傳送給客戶端。這個請求/回應對被視為 API 事務。在使用 Deploy API 之前，您應該熟悉可用於控制請求的輸入變數以及回應輸出的內容。

控制 API 要求的輸入變數

您可以透過在 HTTP 請求中設定的參數來控制 API 呼叫的處理方式。

請求標頭

您必須在 HTTP 請求中包含多個標頭，包括：

- content-type 如果請求正文包含 JSON，則此標頭必須設定為 application/json。

- accept 如果回應正文將包含 JSON，則此標頭必須設定為 application/json。
- 授權基本驗證必須使用以 base64 字串編碼的使用者名稱和密碼進行設定。

請求本文

請求本文的內容會因特定呼叫而異。HTTP 請求本文包含下列其中一項：

- 包含輸入變數（例如新叢集的名稱）的 JSON 物件
- 空白

篩選物件

使用 GET 方法發出 API 呼叫時，您可以根據任何屬性限制或篩選傳回的物件。例如，您可以指定要匹配的確切值：

```
<field>=<query value>
```

除了精確匹配之外，還有其他運算子可用於傳回一組值範圍內的物件。ONTAP Select 支援以下所示的篩選運算子。

操作員	說明
=	等於
<	少於
>	大於
⇐	小於或等於
>=	大於或等於
,	或
!	不等於
*	貪婪萬用字元

您也可以透過在查詢中使用 null 關鍵字或其否定（!null）來根據特定欄位是否已設定傳回一組物件。

選取物件欄位

根據預設、使用 GET 發出 API 呼叫只會傳回唯一識別物件的屬性。這組最小欄位可做為每個物件的金鑰、並會因物件類型而異。您可以使用 fields 查詢參數、以下列方式選取其他物件內容：

- 低成本欄位 指定 `fields=*` 以擷取儲存在本機伺服器記憶體中或存取時處理量很少的物件欄位。
- 昂貴的欄位 指定 `fields=**` 以擷取所有物件欄位，包括需要額外伺服器處理才能存取的欄位。
- 自訂欄位選擇使用 `fields=FIELDNAME`指定所需的確切欄位。請求多個欄位時，值之間必須用逗號分隔，且不能有空格。

最佳實務做法是，您應該始終明確所需欄位。僅在需要時才擷取低成本或高成本欄位集。低成本和高成本的分類是由 NetApp 根據內部效能分析確定。特定欄位的分類可能隨時變更。

對輸出集中的物件進行排序

資源集合中的記錄將按照物件定義的預設順序傳回。您可以使用 order_by 查詢參數，並指定欄位名稱和排序方向來變更順序，如下所示：

```
order_by=<field name> asc|desc
```

例如，您可以先按降序排列類型欄位，再按升序排列 id 欄位：

```
order_by=type desc, id asc
```

當包含多個參數時，必須用逗號分隔各個欄位。

分頁

使用 GET 發出 API 呼叫以存取相同類型的物件集合時，預設會傳回所有相符的物件。如有需要，您可以使用請求中的 max_records 查詢參數來限制傳回的記錄數。例如：

```
max_records=20
```

如有需要，您可以將此參數與其他查詢參數結合使用，以縮小結果集。例如，下列查詢將傳回指定時間之後產生的最多 10 個系統事件：

```
time⇒ 2019-04-04T15:41:29.140265Z&max_records=10
```

您可以發出多個請求來分頁瀏覽事件（或任何物件類型）。每次後續的 API 呼叫都應使用基於上一個結果集中最新事件的新時間值。

解讀 **API** 回應

每個 API 請求都會向用戶端傳回一個回應。您可以檢查該回應以確定請求是否成功，並根據需要擷取其他資料。

HTTP 狀態碼

以下說明 Deploy REST API 所使用的 HTTP 狀態碼。

程式碼	意義	說明
200	確定	表示不會建立新物件的呼叫成功。
201	已建立	已成功建立物件；位置回應標頭包含物件的唯一識別碼。
202	已接受	已啟動長時間執行的背景工作來執行要求，但作業尚未完成。
400	錯誤的請求	要求輸入無法辨識或不適當。
403	禁止	由於授權錯誤，存取被拒絕。
404	找不到	請求中引用的資源不存在。
405	不允許使用此方法	此資源不支援請求中的 HTTP 謂詞。
409	衝突	嘗試建立物件失敗，因為該物件已存在。
500	內部錯誤	伺服器發生一般性內部錯誤。
501	未實作	URI 已知，但無法執行請求。

回應標頭

Deploy 伺服器產生的 HTTP 回應中包含多個標頭、包括：

- request-id 每個成功的 API 請求都會被指派一個唯一的請求識別碼。
- location 當建立物件時，location 標頭包含新物件的完整 URL，包括唯一的物件識別碼。

回應本文

與 API 請求相關聯的回應內容會根據物件、處理類型以及請求成功或失敗而有所不同。回應本文以 JSON 格式呈現。

- 單一物件：可以根據請求傳回包含一組欄位的單一物件。例如，您可以使用 GET 請求，透過唯一識別碼檢索叢集的特定屬性。
- 多個物件可以從資源集合中傳回多個物件。在所有情況下，都使用一致的格式，其中 `num_records`表示記錄數，而記錄包含物件實例的陣列。例如，您可以擷取特定叢集中定義的所有節點。
- Job 物件如果 API 呼叫是非同步處理的，則會傳回一個 Job 物件，該物件用於錨定背景工作。例如，用於部署叢集的 POST 要求是非同步處理的，並傳回一個 Job 物件。
- Error 物件如果發生錯誤，則會始終傳回 Error 物件。例如，嘗試建立名稱已存在的叢集時，將會收到錯誤。
- 空白在某些情況下，不會傳回任何資料，且回應本文為空白。例如，使用 DELETE 刪除現有主機後，回應本文為空白。

使用 **ONTAP Select** 的 **Job** 物件進行非同步處理

某些部署 API 調用，特別是建立或修改資源的調用，可能比其他調用耗時更長。ONTAP Select Deploy 會非同步處理這些耗時較長的請求。

使用 **Job** 物件描述的非同步請求

在進行非同步執行的 API 呼叫後，HTTP 回應代碼 202 表示請求已成功驗證並被接受，但尚未完成。此請求會作為背景工作進行處理，在向用戶端發出初始 HTTP 回應後仍會繼續執行。回應中包含錨定該請求的 Job 物件，以及其唯一識別碼。

您應該參考 ONTAP Select Deploy 線上文件頁面，以確定哪些 API 呼叫是非同步運作的。

查詢與 **API** 請求相關聯的 **Job** 物件

HTTP 回應中傳回的 Job 物件包含多個屬性。您可以查詢 state 屬性來決定請求是否成功完成。Job 物件可以處於以下狀態之一：

- 已排入佇列
- 執行中
- 成功
- 失敗

在輪詢 Job 物件以偵測任務的終止狀態（成功或失敗）時，可以使用兩種技術：

- 標準輪詢請求會立即傳回目前 Job 狀態。
- 長輪詢請求工作狀態僅在發生下列其中一種情況時傳回：
  - 狀態變更時間比輪詢請求中提供的日期時間值更近。
  - 逾時值已過期（1 至 120 秒）

標準輪詢和長輪詢使用相同的 API 呼叫來查詢 Job 物件。但是，長輪詢請求包含兩個查詢參數：`poll_timeout` 和 `last_modified`。

您應該一律使用長輪詢，以減少 Deploy 虛擬機器的工作負載。

發出非同步請求的一般程序

您可以使用下列高階程序來完成非同步 API 呼叫：

1. 發出非同步 API 呼叫。
2. 收到 HTTP 回應 202，表示請求已成功接受。
3. 從回應正文中提取 Job 物件的識別碼。
4. 在一個迴圈中，在每個週期內執行以下操作：
   a. 透過長輪詢請求取得工作的目前狀態
   b. 如果工作處於非終止狀態（已排入佇列、執行中），請再次執行迴圈。
5. 當工作達到終止狀態（成功、失敗）時停止。

使用瀏覽器存取

在使用瀏覽器存取 **ONTAP Select Deploy API** 之前

在使用 Deploy 線上文件頁面之前，您應該注意以下幾件事。

部署計畫

如果您打算在執行特定部署或管理工作時發出 API 呼叫，則應考慮建立部署計畫。這些計畫可以是正式的，也可以是非正式的，通常包含您的目標和要使用的 API 呼叫。如需更多資訊，請參閱使用 Deploy REST API 的工作流程程序。

JSON 範例和參數定義

每個 API 呼叫都會在文件頁面上以統一的格式進行描述。內容包括實作說明、查詢參數和 HTTP 狀態碼。此外、您還可以如下顯示 API 請求和回應中使用的 JSON 的詳細資訊：

- 範例值如果您點擊 API 呼叫中的_範例值_，則會顯示該呼叫的典型 JSON 結構。您可以根據需要修改範例，並將其用作請求的輸入。
- Model 如果您按一下 *Model*，則會顯示完整的 JSON 參數清單，以及每個參數的說明。

發出 **API** 呼叫時需謹慎

您使用 Deploy 文件頁面執行的所有 API 作業都是即時作業。您應注意不要誤建立、更新或刪除組態或其他資料。

存取 **ONTAP Select Deploy** 文件頁面

您必須存取 ONTAP Select Deploy 線上文件頁面才能顯示 API 文件，以及手動發出 API 呼叫。

開始之前

您必須具備以下條件：

- ONTAP Select Deploy 虛擬機器的 IP 位址或網域名稱
- 管理員的使用者名稱和密碼

步驟

1. 在瀏覽器中輸入 URL，然後按 **Enter**：

   ```
   https://<ip_address>/api/ui
   ```

2. 使用管理員使用者名稱和密碼 Sign in。

結果

Deploy 文件網頁會顯示在頁面底部按類別組織的呼叫。

瞭解並執行 **ONTAP Select Deploy API** 呼叫

所有 API 呼叫的詳細資訊均以統一格式記錄並顯示在 ONTAP Select Deploy 線上文件網頁上。透過理解單一 API 呼叫、您可以存取並解讀所有 API 呼叫的詳細資訊。

開始之前

您必須登入 ONTAP Select Deploy 線上文件網頁。您必須擁有建立 ONTAP Select 叢集時指派給該叢集的唯一識別碼。

關於此任務

您可以使用 ONTAP Select 叢集的唯一識別碼擷取描述該叢集的組態資訊。在此範例中，所有歸類為低成本的欄位都會傳回。不過，最佳實務做法是僅要求所需的特定欄位。

步驟

1. 在主頁上，捲動至底部並按一下 **Cluster** 。
2. 按一下 **GET /clusters/{cluster_id}** 以顯示用於傳回 ONTAP Select 叢集相關資訊的 API 呼叫詳細資料。

工作流程處理程序

在使用 **ONTAP Select Deploy API** 工作流程之前

您應該準備好檢閱並使用工作流程程序。

了解工作流程中使用的 **API** 呼叫

ONTAP Select 線上文件頁面包含每個 REST API 呼叫的詳細資訊。此處不再贅述這些詳細資訊，工作流程範例中使用的每個 API 呼叫僅包含您在文件頁面中找到該呼叫所需的資訊。找到特定 API 呼叫後，您可以檢閱該呼叫的完整詳細資訊，包括輸入參數、輸出格式、HTTP 狀態代碼和請求處理類型。

工作流程中每個 API 呼叫都包含下列資訊，以協助您在文件頁面上找到該呼叫：

- 類別 API 呼叫會依功能相關區域或類別在文件頁面上進行組織。若要找到特定的 API 呼叫，請捲動至頁面底部並點擊適用的 API 類別。
- HTTP 動詞 HTTP 動詞可識別對資源執行的動作。每個 API 呼叫都透過單一 HTTP 動詞執行。
- 路徑：路徑決定了在執行呼叫時動作所套用的特定資源。路徑字串會附加到核心 URL 後，形成識別資源的完整 URL。

建立一個 **URL** 以直接存取 **REST API**

除了 ONTAP Select 文件頁面之外，您還可以透過 Python 等程式語言直接存取 Deploy REST API。在這種情況下，核心 URL 與存取線上文件頁面時使用的 URL 略有不同。直接存取 API 時，必須在網域名稱和連接埠字串後附加 /api。例如：

```
http://deploy.mycompany.com/api
```

工作流程 **1**：在 **ESXi** 上建立 **ONTAP Select** 單節點評估叢集

您可以在由 vCenter 管理的 VMware ESXi 主機上部署單節點 ONTAP Select 叢集。叢集是使用評估授權建立的。

在以下情況下，叢集建立工作流程有所不同：

- ESXi 主機不受 vCenter 管理（獨立主機）
- 授權產品的硬體支援
- 叢集已部署在生產環境中，並已購買授權
- 使用 KVM Hypervisor 取代 VMware ESXi

1.註冊 **vCenter** 伺服器憑證

部署到由 vCenter 伺服器管理的 ESXi 主機時，必須先新增憑證，然後再註冊主機。之後，Deploy 管理公用程式可以使用該憑證進行 vCenter 驗證。

類別	**HTTP** 動詞	路徑
部署	POST	/security/credentials

Curl

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step01 'https://10.21.191.150/api/security/credentials'
```

JSON 輸入（步驟 **01**）

```
{
  "hostname": "vcenter.company-demo.com",
  "type": "vcenter",
  "username": "misteradmin@vsphere.local",
  "password": "mypassword"
}
```

處理類型

非同步

輸出

- 位置回應標頭中的認證 ID
- 工作物件

2.註冊 **hypervisor** 主機

您必須新增一個虛擬機器管理程式主機，其中包含 ONTAP Select 節點的虛擬機器將在該主機上執行。

類別	**HTTP** 動詞	路徑
叢集	POST	/hosts

Curl

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step02 'https://10.21.191.150/api/hosts'
```

JSON 輸入（步驟 **02**）

```
{
  "hosts": [
    {
      "hypervisor_type": "ESX",
      "management_server": "vcenter.company-demo.com",
      "name": "esx1.company-demo.com"
    }
  ]
}
```

處理類型

非同步

輸出

- 位置回應標頭中的主機 ID
- 工作物件

3.建立叢集

建立 ONTAP Select 叢集時，基本叢集配置會被註冊，節點名稱也會由 Deploy 自動產生。

類別	**HTTP** 動詞	路徑
叢集	POST	/clusters

Curl

對於單節點叢集，查詢參數 node_count 應設定為 1。

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step03 'https://10.21.191.150/api/clusters? node_count=1'
```

JSON 輸入（步驟 **03**）

```
{
  "name": "my_cluster"
}
```

處理類型

同步

輸出

- 位置回應標頭中的叢集 ID

4.設定叢集

在配置叢集時，您必須提供幾個屬性。

類別	**HTTP** 動詞	路徑
叢集	PATCH	/clusters/{cluster_id}

Curl

您必須提供叢集 ID。

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step04 'https://10.21.191.150/api/clusters/CLUSTERID'
```

JSON 輸入（步驟 04）

```
{
  "dns_info": {
    "domains": ["lab1.company-demo.com"],
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
    },
    "ontap_image_version": "9.5",
    "gateway": "10.206.80.1",
    "ip": "10.206.80.115",
    "netmask": "255.255.255.192",
    "ntp_servers": {"10.206.80.183"}
}
```

處理類型

同步

輸出

無

5.擷取節點名稱

Deploy 管理公用程式會在建立叢集時自動產生節點識別碼和名稱。在配置節點之前，必須先取得已指派的 ID。

類別	**HTTP** 動詞	路徑
叢集	取得	/clusters/{cluster_id}/nodes

Curl

您必須提供叢集 ID。

```
curl -iX GET -u admin:<password> -k
'https://10.21.191.150/api/clusters/CLUSTERID/nodes?fields=id,name'
```

處理類型

同步

輸出

- 陣列記錄，每筆記錄描述一個具有唯一 ID 和名稱的單一節點

6.配置節點

您必須提供節點的基本組態，這是用於組態節點的三個 API 呼叫中的第一個。

類別	HTTP 動詞	路徑
叢集	路徑	/clusters/{cluster_id}/nodes/{node_id}

Curl

您必須提供叢集 ID 和節點 ID。

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step06 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID'
```

JSON 輸入（步驟 **06**）

您必須提供 ONTAP Select 節點將運行的主機 ID。

```
{
  "host": {
    "id": "HOSTID"
    },
  "instance_type": "small",
  "ip": "10.206.80.101",
  "passthrough_disks": false
}
```

處理類型

同步

輸出

無

7.擷取節點網路

您必須識別單節點叢集中節點所使用的資料網路和管理網路。單節點叢集不使用內部網路。

類別	HTTP 動詞	路徑
叢集	取得	/clusters/{cluster_id}/nodes/{node_id}/networks

Curl

您必須提供叢集 ID 和節點 ID。

```
curl -iX GET -u admin:<password> -k 'https://10.21.191.150/api/
clusters/CLUSTERID/nodes/NODEID/networks?fields=id,purpose'
```

處理類型

同步

輸出

- 包含兩筆記錄的陣列，每筆記錄描述節點的單一網路，包括唯一 ID 和用途

8.設定節點網路

您必須設定資料網路和管理網路。單節點叢集不使用內部網路。

對每個網路分別發出以下 API 呼叫兩次。

類別	**HTTP** 動詞	路徑
叢集	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

您必須提供叢集 ID、節點 ID 和網路 ID。

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step08 'https://10.21.191.150/api/clusters/
CLUSTERID/nodes/NODEID/networks/NETWORKID'
```

JSON 輸入（步驟 **08**）

您需要提供網路名稱。

```
{
  "name": "sDOT_Network"
}
```

處理類型

同步

輸出

無

9.設定節點儲存資源池

配置節點的最後一步是附加儲存池。您可以透過 vSphere Web 用戶端或部署 REST API（選用）來確定可用的儲存池。

類別	**HTTP** 動詞	路徑
叢集	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

您必須提供叢集 ID、節點 ID 和網路 ID。

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step09 'https://10.21.191.150/api/clusters/ CLUSTERID/nodes/NODEID'
```

JSON 輸入（步驟 09）

儲存池容量為 2 TB。

```
{
  "pool_array": [
    {
      "name": "sDOT-01",
      "capacity": 2147483648000
    }
  ]
}
```

處理類型

同步

輸出

無

10.部署叢集

叢集和節點配置完成後，即可部署叢集。

類別	**HTTP** 動詞	路徑
叢集	POST	/clusters/{cluster_id}/deploy

Curl

您必須提供叢集 ID。

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step10 'https://10.21.191.150/api/clusters/CLUSTERID/deploy'
```

JSON 輸入（步驟 10）

您必須提供 ONTAP 管理員帳戶的密碼。

```
{
  "ontap_credentials": {
    "password": "mypassword"
  }
}
```

處理類型

非同步

輸出

- 工作物件

相關資訊

"部署一個為期 90 天的 ONTAP Select 叢集評估執行個體"

使用 Python 存取

在使用 Python 存取 ONTAP Select Deploy API 之前

在執行範例 Python 指令碼之前、您必須先準備好環境。

在執行 Python 指令碼之前，必須確保環境配置正確：

- 必須安裝最新適用的 Python2 版本。範例程式碼已使用 Python2 進行測試。它們也應該可以移植到 Python3，但尚未進行相容性測試。
- 必須安裝 Requests 和 urllib3 函式庫。您可以根據您的環境使用 pip 或其他 Python 管理工具。
- 執行指令碼的用戶端工作站必須能夠透過網路存取 ONTAP Select Deploy 虛擬機器。

此外、您必須擁有下列資訊：

- Deploy 虛擬機器的 IP 位址
- Deploy 管理員帳戶的使用者名稱和密碼

了解 ONTAP Select Deploy 的 Python 腳本

範例 Python 指令碼可讓您執行多種不同的工作。在即時 Deploy 執行個體使用這些指令碼之前、您應該先瞭解這些指令碼。

常見設計特性

這些指令碼的設計具有下列共同特性：

- 在用戶端機器上從命令列介面執行您可以從任何正確設定的用戶端機器執行 Python 指令碼。如需詳細資訊、請參閱 *Before you begin* 。
- 接受 CLI 輸入參數每個指令碼都透過 CLI 的輸入參數進行控制。

- 讀取輸入檔案每個指令碼都會根據其用途讀取輸入檔案。建立或刪除叢集時、您必須提供 JSON 組態檔。新增節點授權時、您必須提供有效的授權檔。
- 使用通用支援模組通用支援模組 *deploy_requests.py* 包含一個類別。每個腳本都會匯入並使用它。

建立叢集

您可以使用 cluster.py 指令碼建立 ONTAP Select 叢集。根據 CLI 參數和 JSON 輸入檔案的內容、您可以修改指令碼以適應部署環境、如下所示：

- Hypervisor 您可以部署至 ESXi 或 KVM（取決於 Deploy 版本）。部署至 ESXi 時,Hypervisor 可由 vCenter 管理,也可以是獨立主機。
- 叢集大小 您可以部署單節點或多節點叢集。
- 評估或正式作業授權 您可以使用評估或購買的正式作業授權來部署叢集。

指令碼的 CLI 輸入參數包括：

- Deploy 伺服器的主機名稱或 IP 位址
- 管理使用者帳戶的密碼
- JSON 設定檔案名稱
- 訊息輸出的詳細資訊旗標

新增節點授權

如果您選擇部署正式作業叢集、則必須使用指令碼 *add_license.py* 為每個節點新增授權。您可以在部署叢集之前或之後新增授權。

指令碼的 CLI 輸入參數包括：

- Deploy 伺服器的主機名稱或 IP 位址
- 管理使用者帳戶的密碼
- 授權檔案的名稱
- 具有新增授權權限的 ONTAP 使用者名稱
- ONTAP 使用者的密碼

刪除叢集

您可以使用腳本 *delete_cluster.py* 刪除現有的 ONTAP Select 叢集。

指令碼的 CLI 輸入參數包括：

- Deploy 伺服器的主機名稱或 IP 位址
- 管理使用者帳戶的密碼
- JSON 設定檔案名稱

Python 程式碼範例

用於建立 ONTAP Select 叢集的指令碼

您可以使用以下腳本，根據腳本中定義的參數和 JSON 輸入檔案建立叢集。

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: cluster.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import traceback
import argparse
import json
import logging

from deploy_requests import DeployRequests


def add_vcenter_credentials(deploy, config):
    """ Add credentials for the vcenter if present in the config """
    log_debug_trace()

    vcenter = config.get('vcenter', None)
    if vcenter and not deploy.resource_exists('/security/credentials',
                                              'hostname', vcenter[
'hostname']):
        log_info("Registering vcenter {} credentials".format(vcenter[
'hostname']))
        data = {k: vcenter[k] for k in ['hostname', 'username', 'password
']}
```

```
        data['type'] = "vcenter"
        deploy.post('/security/credentials', data)


def add_standalone_host_credentials(deploy, config):
    """ Add credentials for standalone hosts if present in the config.
        Does nothing if the host credential already exists on the Deploy.
    """
    log_debug_trace()

    hosts = config.get('hosts', [])
    for host in hosts:
        # The presense of the 'password' will be used only for standalone
hosts.
        # If this host is managed by a vcenter, it should not have a host
'password' in the json.
        if 'password' in host and not deploy.resource_exists(
'/security/credentials',
                                                             'hostname',
host['name']):
            log_info("Registering host {} credentials".format(host['name
']))
            data = {'hostname': host['name'], 'type': 'host',
                    'username': host['username'], 'password': host[
'password']}
            deploy.post('/security/credentials', data)


def register_unkown_hosts(deploy, config):
    ''' Registers all hosts with the deploy server.
        The host details are read from the cluster config json file.

        This method will skip any hosts that are already registered.
        This method will exit the script if no hosts are found in the
config.
    '''
    log_debug_trace()

    data = {"hosts": []}
    if 'hosts' not in config or not config['hosts']:
        log_and_exit("The cluster config requires at least 1 entry in the
'hosts' list got {}".format(config))

    missing_host_cnt = 0
    for host in config['hosts']:
        if not deploy.resource_exists('/hosts', 'name', host['name']):
```

```
            missing_host_cnt += 1
            host_config = {"name": host['name'], "hypervisor_type": host[
'type']}
            if 'mgmt_server' in host:
                host_config["management_server"] = host['mgmt_server']
                log_info(
                   "Registering from vcenter {mgmt_server}".format(**
host))

            if 'password' in host and 'user' in host:
                host_config['credential'] = {
                    "password": host['password'], "username": host['user
']}

            log_info("Registering {type} host {name}".format(**host))
            data["hosts"].append(host_config)

    # only post /hosts if some missing hosts were found
    if missing_host_cnt:
        deploy.post('/hosts', data, wait_for_job=True)


def add_cluster_attributes(deploy, config):
    ''' POST a new cluster with all needed attribute values.
        Returns the cluster_id of the new config
    '''
    log_debug_trace()

    cluster_config = config['cluster']
    cluster_id = deploy.find_resource('/clusters', 'name', cluster_config
['name'])

    if not cluster_id:
        log_info("Creating cluster config named {name}".format(
**cluster_config))

        # Filter to only the valid attributes, ignores anything else in
the json
        data = {k: cluster_config[k] for k in [
            'name', 'ip', 'gateway', 'netmask', 'ontap_image_version',
'dns_info', 'ntp_servers']}

        num_nodes = len(config['nodes'])

        log_info("Cluster properties: {}".format(data))

        resp = deploy.post('/v3/clusters?node_count={}'.format(num_nodes),
```

```
data)
        cluster_id = resp.headers.get('Location').split('/')[-1]

    return cluster_id


def get_node_ids(deploy, cluster_id):
    ''' Get the the ids of the nodes in a cluster. Returns a list of
node_ids.'''
    log_debug_trace()

    response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
    node_ids = [node['id'] for node in response.json().get('records')]
    return node_ids


def add_node_attributes(deploy, cluster_id, node_id, node):
    ''' Set all the needed properties on a node '''
    log_debug_trace()

    log_info("Adding node '{}' properties".format(node_id))

    data = {k: node[k] for k in ['ip', 'serial_number', 'instance_type',
                                 'is_storage_efficiency_enabled'] if k in
node}
    # Optional: Set a serial_number
    if 'license' in node:
        data['license'] = {'id': node['license']}

    # Assign the host
    host_id = deploy.find_resource('/hosts', 'name', node['host_name'])
    if not host_id:
        log_and_exit("Host names must match in the 'hosts' array, and the
nodes.host_name property")

    data['host'] = {'id': host_id}

    # Set the correct raid_type
    is_hw_raid = not node['storage'].get('disks')  # The presence of a
list of disks indicates sw_raid
    data['passthrough_disks'] = not is_hw_raid

    # Optionally set a custom node name
    if 'name' in node:
        data['name'] = node['name']

    log_info("Node properties: {}".format(data))
```

```
    deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id),
data)


def add_node_networks(deploy, cluster_id, node_id, node):
    ''' Set the network information for a node '''
    log_debug_trace()

    log_info("Adding node '{}' network properties".format(node_id))

    num_nodes = deploy.get_num_records('/clusters/{}/nodes'.format
(cluster_id))

    for network in node['networks']:

        # single node clusters do not use the 'internal' network
        if num_nodes == 1 and network['purpose'] == 'internal':
            continue

        # Deduce the network id given the purpose for each entry
        network_id = deploy.find_resource('/clusters/{}/nodes/{}/networks
'.format(cluster_id, node_id),
                                          'purpose', network['purpose'])
        data = {"name": network['name']}
        if 'vlan' in network and network['vlan']:
            data['vlan_id'] = network['vlan']

        deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format(
cluster_id, node_id, network_id), data)


def add_node_storage(deploy, cluster_id, node_id, node):
    ''' Set all the storage information on a node '''
    log_debug_trace()

    log_info("Adding node '{}' storage properties".format(node_id))
    log_info("Node storage: {}".format(node['storage']['pools']))

    data = {'pool_array': node['storage']['pools']}  # use all the json
properties
    deploy.post(
        '/clusters/{}/nodes/{}/storage/pools'.format(cluster_id, node_id),
data)

    if 'disks' in node['storage'] and node['storage']['disks']:
        data = {'disks': node['storage']['disks']}
        deploy.post(
```

```
            '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id,
node_id), data)


def create_cluster_config(deploy, config):
    ''' Construct a cluster config in the deploy server using the input
json data '''
    log_debug_trace()

    cluster_id = add_cluster_attributes(deploy, config)

    node_ids = get_node_ids(deploy, cluster_id)
    node_configs = config['nodes']

    for node_id, node_config in zip(node_ids, node_configs):
        add_node_attributes(deploy, cluster_id, node_id, node_config)
        add_node_networks(deploy, cluster_id, node_id, node_config)
        add_node_storage(deploy, cluster_id, node_id, node_config)

    return cluster_id


def deploy_cluster(deploy, cluster_id, config):
    ''' Deploy the cluster config to create the ONTAP Select VMs. '''
    log_debug_trace()
    log_info("Deploying cluster: {}".format(cluster_id))

    data = {'ontap_credential': {'password': config['cluster'][
'ontap_admin_password']}}
    deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format
(cluster_id),
                data, wait_for_job=True)


def log_debug_trace():
    stack = traceback.extract_stack()
    parent_function = stack[-2][2]
    logging.getLogger('deploy').debug('Calling %s()' % parent_function)


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)
```

```
def configure_logging(verbose):
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    if verbose:
        logging.basicConfig(level=logging.DEBUG, format=FORMAT)
    else:
        logging.basicConfig(level=logging.INFO, format=FORMAT)
        logging.getLogger('requests.packages.urllib3.connectionpool'
).setLevel(
            logging.WARNING)


def main(args):
    configure_logging(args.verbose)
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)

        add_vcenter_credentials(deploy, config)

        add_standalone_host_credentials(deploy, config)

        register_unkown_hosts(deploy, config)

        cluster_id = create_cluster_config(deploy, config)

        deploy_cluster(deploy, cluster_id, config)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to construct and deploy a cluster.')
    parser.add_argument('-d', '--deploy', help='Hostname or IP address of
Deploy server')
    parser.add_argument('-p', '--password', help='Admin password of Deploy
server')
    parser.add_argument('-c', '--config_file', help='Filename of the
cluster config')
    parser.add_argument('-v', '--verbose', help='Display extra debugging
messages for seeing exact API calls and responses',
                        action='store_true', default=False)
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
```

```
    main(args)
```

用於建立 ONTAP Select 叢集的指令碼 JSON

使用 Python 程式碼範例建立或刪除 ONTAP Select 叢集時，必須提供一個 JSON 檔案作為腳本的輸入。您可以根據部署計劃複製並修改對應的 JSON 範例。

ESXi 上的單節點叢集

```
{
  "hosts": [
    {
      "password": "mypassword1",
      "name": "host-1234",
      "type": "ESX",
      "username": "admin"
    }
  ],

  "cluster": {
    "dns_info": {
      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
        "lab3.company-demo.com", "lab4.company-demo.com"
        ],

      "dns_ips": ["10.206.80.135", "10.206.80.136"]
      },
      "ontap_image_version": "9.7",
      "gateway": "10.206.80.1",
      "ip": "10.206.80.115",
      "name": "mycluster",
      "ntp_servers": ["10.206.80.183", "10.206.80.142"],
      "ontap_admin_password": "mypassword2",
      "netmask": "255.255.254.0"
  },

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",
      "name": "node-1",
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
```

```
            "vlan": 1234
          },
          {
            "name": "ontap-external",
            "purpose": "data",
            "vlan": null
          },
          {
            "name": "ontap-internal",
            "purpose": "internal",
            "vlan": null
          }
        ],
        "host_name": "host-1234",
        "is_storage_efficiency_enabled": false,
        "instance_type": "small",
        "storage": {
          "disk": [],
          "pools": [
            {
              "name": "storage-pool-1",
              "capacity": 4802666790125
            }
          ]
        }
      }
    ]
}
```

在 **ESXi** 上使用 **vCenter** 的單節點叢集

```
{
  "hosts": [
    {
      "name":"host-1234",
      "type":"ESX",
      "mgmt_server":"vcenter-1234"
    }
  ],

  "cluster": {
    "dns_info": {"domains": ["lab1.company-demo.com", "lab2.company-
demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
      ],
```

```
    "dns_ips": ["10.206.80.135","10.206.80.136"]
  },

  "ontap_image_version":"9.7",
  "gateway":"10.206.80.1",
  "ip":"10.206.80.115",
  "name":"mycluster",
  "ntp_servers": ["10.206.80.183","10.206.80.142"],
  "ontap_admin_password":"mypassword2",
  "netmask":"255.255.254.0"
  },

  "vcenter": {
    "password":"mypassword2",
    "hostname":"vcenter-1234",
    "username":"selectadmin"
  },

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip":"10.206.80.114",
      "name":"node-1",
      "networks": [
        {
          "name":"ONTAP-Management",
          "purpose":"mgmt",
          "vlan":null
        },
        {
          "name": "ONTAP-External",
          "purpose":"data",
          "vlan":null
        },
        {
          "name": "ONTAP-Internal",
          "purpose":"internal",
          "vlan":null
        }
      ],

      "host_name": "host-1234",
      "is_storage_efficiency_enabled": false,
      "instance_type": "small",
      "storage": {
        "disk":[],
```

```
        "pools": [
          {
            "name": "storage-pool-1",
            "capacity":5685190380748
          }
        ]
      }
    }
  ]
}
```

KVM 上的單節點叢集

```
{
  "hosts": [
    {
      "password": "mypassword1",
      "name":"host-1234",
      "type":"KVM",
      "username":"root"
    }
  ],

  "cluster": {
    "dns_info": {
      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
        "lab3.company-demo.com", "lab4.company-demo.com"
      ],

      "dns_ips": ["10.206.80.135", "10.206.80.136"]
    },

    "ontap_image_version": "9.7",
    "gateway":"10.206.80.1",
    "ip":"10.206.80.115",
    "name":"CBF4ED97",
    "ntp_servers": ["10.206.80.183", "10.206.80.142"],
    "ontap_admin_password": "mypassword2",
    "netmask":"255.255.254.0"
  },
  "nodes": [
    {
      "serial_number":"3200000nn",
      "ip":"10.206.80.115",
      "name": "node-1",
```

```
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
          "vlan":1234
        },
        {
          "name": "ontap-external",
          "purpose": "data",
          "vlan": null
        },
        {
          "name": "ontap-internal",
          "purpose": "internal",
          "vlan": null
        }
      ],

      "host_name": "host-1234",
      "is_storage_efficiency_enabled": false,
      "instance_type": "small",
      "storage": {
        "disk": [],
        "pools": [
          {
            "name": "storage-pool-1",
            "capacity": 4802666790125
          }
        ]
      }
    }
  ]
}
```

用於新增 **ONTAP Select** 節點授權的指令碼

您可以使用以下腳本為 ONTAP Select 節點新增授權。

```
#!/usr/bin/env python
##-----------------------------------------------------------------------------
#
# File: add_license.py
#
# (C) Copyright 2019 NetApp, Inc.
#
```

```
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##------------------------------------------------------------------

import argparse
import logging
import json

from deploy_requests import DeployRequests


def post_new_license(deploy, license_filename):
    log_info('Posting a new license: {}'.format(license_filename))

    # Stream the file as multipart/form-data
    deploy.post('/licensing/licenses', data={},
                files={'license_file': open(license_filename, 'rb')})

    # Alternative if the NLF license data is converted to a string.
    # with open(license_filename, 'rb') as f:
    #    nlf_data = f.read()
    #    r = deploy.post('/licensing/licenses', data={},
    #                    files={'license_file': (license_filename,
nlf_data)})


def put_license(deploy, serial_number, data, files):
    log_info('Adding license for serial number: {}'.format(serial_number))

    deploy.put('/licensing/licenses/{}'.format(serial_number), data=data,
files=files)


def put_used_license(deploy, serial_number, license_filename,
ontap_username, ontap_password):
    ''' If the license is used by an 'online' cluster, a username/password
must be given. '''

    data = {'ontap_username': ontap_username, 'ontap_password':
```

```
ontap_password}
    files = {'license_file': open(license_filename, 'rb')}

    put_license(deploy, serial_number, data, files)


def put_free_license(deploy, serial_number, license_filename):
    data = {}
    files = {'license_file': open(license_filename, 'rb')}

    put_license(deploy, serial_number, data, files)


def get_serial_number_from_license(license_filename):
    ''' Read the NLF file to extract the serial number '''
    with open(license_filename) as f:
        data = json.load(f)

        statusResp = data.get('statusResp', {})
        serialNumber = statusResp.get('serialNumber')
        if not serialNumber:
            log_and_exit("The license file seems to be missing the
serialNumber")

        return serialNumber


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)


def configure_logging():
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    logging.basicConfig(level=logging.INFO, format=FORMAT)
    logging.getLogger('requests.packages.urllib3.connectionpool').
setLevel(logging.WARNING)


def main(args):
    configure_logging()
    serial_number = get_serial_number_from_license(args.license)
```

```
    deploy = DeployRequests(args.deploy, args.password)

    # First check if there is already a license resource for this serial-
number
    if deploy.find_resource('/licensing/licenses', 'id', serial_number):

        # If the license already exists in the Deploy server, determine if
its used
        if deploy.find_resource('/clusters', 'nodes.serial_number',
serial_number):

            # In this case, requires ONTAP creds to push the license to
the node
            if args.ontap_username and args.ontap_password:
                put_used_license(deploy, serial_number, args.license,
                                 args.ontap_username, args.ontap_password)
            else:
                print("ERROR: The serial number for this license is in
use. Please provide ONTAP credentials.")
        else:
            # License exists, but its not used
            put_free_license(deploy, serial_number, args.license)
    else:
        # No license exists, so register a new one as an available license
for later use
        post_new_license(deploy, args.license)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to add or update a new or used NLF license file.')
    parser.add_argument('-d', '--deploy', required=True, type=str, help=
'Hostname or IP address of ONTAP Select Deploy')
    parser.add_argument('-p', '--password', required=True, type=str, help
='Admin password of Deploy server')
    parser.add_argument('-l', '--license', required=True, type=str, help=
'Filename of the NLF license data')
    parser.add_argument('-u', '--ontap_username', type=str,
                        help='ONTAP Select username with privelege to add
the license. Only provide if the license is used by a Node.')
    parser.add_argument('-o', '--ontap_password', type=str,
                        help='ONTAP Select password for the
ontap_username. Required only if ontap_username is given.')
    return parser.parse_args()

if __name__ == '__main__':
```

```
    args = parseArgs()
    main(args)
```

用於刪除 ONTAP Select 叢集的指令碼

您可以使用以下 CLI 指令碼刪除現有的叢集。

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: delete_cluster.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import argparse
import json
import logging

from deploy_requests import DeployRequests

def find_cluster(deploy, cluster_name):
    return deploy.find_resource('/clusters', 'name', cluster_name)


def offline_cluster(deploy, cluster_id):
    # Test that the cluster is online, otherwise do nothing
    response = deploy.get('/clusters/{}?fields=state'.format(cluster_id))
    cluster_data = response.json()['record']
    if cluster_data['state'] == 'powered_on':
        log_info("Found the cluster to be online, modifying it to be
powered_off.")
        deploy.patch('/clusters/{}'.format(cluster_id), {'availability':
'powered_off'}, True)
```

```
def delete_cluster(deploy, cluster_id):
    log_info("Deleting the cluster({}).".format(cluster_id))
    deploy.delete('/clusters/{}'.format(cluster_id), True)
    pass


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def configure_logging():
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    logging.basicConfig(level=logging.INFO, format=FORMAT)
    logging.getLogger('requests.packages.urllib3.connectionpool').
setLevel(logging.WARNING)


def main(args):
    configure_logging()
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)

        cluster_id = find_cluster(deploy, config['cluster']['name'])

        log_info("Found the cluster {} with id: {}.".format(config[
'cluster']['name'], cluster_id))

        offline_cluster(deploy, cluster_id)

        delete_cluster(deploy, cluster_id)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to delete a cluster')
    parser.add_argument('-d', '--deploy', required=True, type=str, help=
'Hostname or IP address of Deploy server')
    parser.add_argument('-p', '--password', required=True, type=str, help
='Admin password of Deploy server')
    parser.add_argument('-c', '--config_file', required=True, type=str,
help='Filename of the cluster json config')
    return parser.parse_args()
```

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

ONTAP Select 的通用支援 Python 模組

所有 Python 指令碼都使用單一模組中的通用 Python 類別。

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: deploy_requests.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import json
import logging
import requests

requests.packages.urllib3.disable_warnings()

class DeployRequests(object):
    '''
    Wrapper class for requests that simplifies the ONTAP Select Deploy
    path creation and header manipulations for simpler code.
    '''

    def __init__(self, ip, admin_password):
        self.base_url = 'https://{}/api'.format(ip)
        self.auth = ('admin', admin_password)
        self.headers = {'Accept': 'application/json'}
        self.logger = logging.getLogger('deploy')
```

```
    def post(self, path, data, files=None, wait_for_job=False):
        if files:
            self.logger.debug('POST FILES:')
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     files=files)
        else:
            self.logger.debug('POST DATA: %s', data)
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     json=data,
                                     headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def patch(self, path, data, wait_for_job=False):
        self.logger.debug('PATCH DATA: %s', data)
        response = requests.patch(self.base_url + path,
                                  auth=self.auth, verify=False,
                                  json=data,
                                  headers=self.headers)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def put(self, path, data, files=None, wait_for_job=False):
        if files:
            print('PUT FILES: {}'.format(data))
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
                                    data=data,
                                    files=files)
        else:
            self.logger.debug('PUT DATA:')
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
```

```
                                    json=data,
                                    headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def get(self, path):
        """ Get a resource object from the specified path """
        response = requests.get(self.base_url + path, auth=self.auth,
verify=False)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)
        return response

    def delete(self, path, wait_for_job=False):
        """ Delete's a resource from the specified path """
        response = requests.delete(self.base_url + path, auth=self.auth,
verify=False)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def find_resource(self, path, name, value):
        ''' Returns the 'id' of the resource if it exists, otherwise None
'''
        resource = None
        response = self.get('{path}?{field}={value}'.format(
                            path=path, field=name, value=value))
        if response.status_code == 200 and response.json().get(
'num_records') >= 1:
            resource = response.json().get('records')[0].get('id')
        return resource

    def get_num_records(self, path, query=None):
        ''' Returns the number of records found in a container, or None on
error '''
```

```
        resource = None
        query_opt = '?{}'.format(query) if query else ''
        response = self.get('{path}{query}'.format(path=path, query
=query_opt))
        if response.status_code == 200 :
            return response.json().get('num_records')
        return None

    def resource_exists(self, path, name, value):
        return self.find_resource(path, name, value) is not None

    def wait_for_job(self, response, poll_timeout=120):
        last_modified = response['job']['last_modified']
        job_id = response['job']['id']

        self.logger.info('Event: ' + response['job']['message'])

        while True:
            response = self.get('/jobs/{}?fields=state,message&'
                                'poll_timeout={}&last_modified=>={}'
.format(
                                    job_id, poll_timeout, last_modified))

            job_body = response.json().get('record', {})

            # Show interesting message updates
            message = job_body.get('message', '')
            self.logger.info('Event: ' + message)

            # Refresh the last modified time for the poll loop
            last_modified = job_body.get('last_modified')

            # Look for the final states
            state = job_body.get('state', 'unknown')
            if state in ['success', 'failure']:
                if state == 'failure':
                    self.logger.error('FAILED background job.\nJOB: %s',
job_body)
                    exit(1)    # End the script if a failure occurs
                break

    def exit_on_errors(self, response):
        if response.status_code >= 400:
            self.logger.error('FAILED request to URL: %s\nHEADERS: %s
\nRESPONSE BODY: %s',
                              response.request.url,
```

```
                                self.filter_headers(response),
                                response.text)
        response.raise_for_status()    # Displays the response error, and
exits the script

    @staticmethod
    def filter_headers(response):
        ''' Returns a filtered set of the response headers '''
        return {key: response.headers[key] for key in ['Location',
'request-id'] if key in response.headers}
```

用於調整 ONTAP Select 叢集節點大小的指令碼

您可以使用以下腳本來調整 ONTAP Select 叢集中的節點大小。

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: resize_nodes.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import argparse
import logging
import sys

from deploy_requests import DeployRequests


def _parse_args():
    """ Parses the arguments provided on the command line when executing
this
        script and returns the resulting namespace. If all required
```

```
arguments
        are not provided, an error message indicating the mismatch is
printed and
        the script will exit.
    """

    parser = argparse.ArgumentParser(description=(
        'Uses the ONTAP Select Deploy API to resize the nodes in the
cluster.'
        ' For example, you might have a small (4 CPU, 16GB RAM per node) 2
node'
        ' cluster and wish to resize the cluster to medium (8 CPU, 64GB
RAM per'
        ' node). This script will take in the cluster details and then
perform'
        ' the operation and wait for it to complete.'
    ))
    parser.add_argument('--deploy', required=True, help=(
        'Hostname or IP of the ONTAP Select Deploy VM.'
    ))
    parser.add_argument('--deploy-password', required=True, help=(
        'The password for the ONTAP Select Deploy admin user.'
    ))
    parser.add_argument('--cluster', required=True, help=(
        'Hostname or IP of the cluster management interface.'
    ))
    parser.add_argument('--instance-type', required=True, help=(
        'The desired instance size of the nodes after the operation is
complete.'
    ))
    parser.add_argument('--ontap-password', required=True, help=(
        'The password for the ONTAP administrative user account.'
    ))
    parser.add_argument('--ontap-username', default='admin', help=(
        'The username for the ONTAP administrative user account. Default:
admin.'
    ))
    parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME', help=(
        'A space separated list of node names for which the resize
operation'
        ' should be performed. The default is to apply the resize to all
nodes in'
        ' the cluster. If a list of nodes is provided, it must be provided
in HA'
        ' pairs. That is, in a 4 node cluster, nodes 1 and 2 (partners)
must be'
```

```
        ' resized in the same operation.'
    ))
    return parser.parse_args()


def _get_cluster(deploy, parsed_args):
    """ Locate the cluster using the arguments provided """

    cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args
.cluster)
    if not cluster_id:
        return None
    return deploy.get('/clusters/%s?fields=nodes' % cluster_id).json()[
'record']


def _get_request_body(parsed_args, cluster):
    """ Build the request body """

    changes = {'admin_password': parsed_args.ontap_password}

    # if provided, use the list of nodes given, else use all the nodes in
the cluster
    nodes = [node for node in cluster['nodes']]
    if parsed_args.nodes:
        nodes = [node for node in nodes if node['name'] in parsed_args
.nodes]

    changes['nodes'] = [
        {'instance_type': parsed_args.instance_type, 'id': node['id']} for
node in nodes]

    return changes


def main():
    """ Set up the resize operation by gathering the necessary data and
then send
        the request to the ONTAP Select Deploy server.
    """

    logging.basicConfig(
        format='[%(asctime)s] [%(levelname)5s] %(message)s', level=
logging.INFO,)

    logging.getLogger('requests.packages.urllib3').setLevel(logging
.WARNING)
```

```
    parsed_args = _parse_args()
    deploy = DeployRequests(parsed_args.deploy, parsed_args
.deploy_password)

    cluster = _get_cluster(deploy, parsed_args)
    if not cluster:
        deploy.logger.error(
            'Unable to find a cluster with a management IP of %s' %
parsed_args.cluster)
        return 1

    changes = _get_request_body(parsed_args, cluster)
    deploy.patch('/clusters/%s' % cluster['id'], changes, wait_for_job
=True)

if __name__ == '__main__':
    sys.exit(main())
```