

# 使用 VMware vRealize Orchestrator 用戶端

2021 年 11 月 19 日  
vRealize Orchestrator 8.6

您可以在 VMware 網站上找到最新的技術文件，網址如下：

<https://docs.vmware.com/tw/>

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

Copyright © 2008-2021 VMware, Inc. 保留所有權利。版權與商標資訊。

# 目錄

- 1 使用 VMware vRealize Orchestrator 用戶端 6**
- 2 vRealize Orchestrator 用戶端簡介 7**
  - vRealize Orchestrator 用戶端使用量儀表板 8
  - vRealize Orchestrator Client 中的內容組織整理 8
    - 建立資料夾或子資料夾 9
    - 移動物件和資料夾 10
    - 刪除資料夾或子資料夾 10
- 3 設定 vRealize Orchestrator Client 12**
  - vRealize Orchestrator 角色和群組 12
    - 在 vRealize Orchestrator Client 中指派角色 13
    - 在 vRealize Automation 中設定 vRealize Orchestrator 用戶端角色 14
    - 在 vRealize Orchestrator Client 中建立群組 14
  - vRealize Orchestrator 物件的版本歷程記錄 15
    - 將工作流程還原至先前版本 16
    - 工作流程版本之間的視覺比較 16
    - 使用 Git 將 vRealize Orchestrator 內容詳細目錄重設為先前的狀態 17
- 4 vRealize Orchestrator 使用案例 19**
  - 如何使用 Python 在 vRealize Orchestrator 中整合 Amazon Web Services 19
    - 建立初始 Python 指令碼 20
    - 建立 Amazon Web Services 動作 21
    - 偵錯 Amazon Web Services 動作 22
    - 更新 Amazon Web Services 動作 24
  - 如何使用 Git 分支管理 vRealize Orchestrator 物件詳細目錄 25
    - 準備 GitLab 環境 26
    - 設定與 Git 存放庫的連線 26
    - 將變更推送至 Git 存放庫 27
  - 如何使用第三方模組呼叫 vRealize Automation 專案 API 29
    - 建立呼叫 vRealize Automation 專案 API 的 Python 指令碼 30
    - 建立呼叫 vRealize Automation 專案 API 的 Node.js 指令碼 32
    - 建立呼叫 vRealize Automation 專案 API 的 PowerShell 指令碼 34
- 5 管理工作流程 37**
  - vRealize Orchestrator 工作流程程式庫中的標準工作流程 37
  - 建立工作流程 38

從父系工作流程編輯工作流程和動作	38
vRealize Orchestrator 輸入表單設計工具	39
在 vRealize Orchestrator 用戶端中建立工作流程輸入參數對話方塊	39
vRealize Orchestrator 用戶端中的輸入參數內容	40
使用動作驗證 vRealize Orchestrator 工作流程輸入	40
vRealize Orchestrator 用戶端中的使用者互動要求	42
排程工作流程	42
在 vRealize Orchestrator 用戶端中編輯排定的工作	42
在工作流程中尋找物件參考	43
<b>6 管理動作</b>	<b>44</b>
建立動作	44
執行和偵錯動作	45
執行動作	45
偵錯動作	46
Python、Node.js 和 PowerShell 指令碼的核心概念	47
Python、Node.js 和 PowerShell 指令碼的執行階段限制	48
<b>7 管理組態元素</b>	<b>49</b>
建立組態元素	49
<b>8 管理原則</b>	<b>51</b>
建立和套用原則	51
原則元素	52
管理原則執行	52
<b>9 管理資源元素</b>	<b>54</b>
<b>10 管理套件</b>	<b>55</b>
建立套件	55
匯出套件	56
匯入套件	57
<b>11 vRealize Orchestrator 用戶端中的疑難排解</b>	<b>58</b>
vRealize Orchestrator 用戶端中的度量資料	58
在 vRealize Orchestrator 用戶端中剖析工作流程	58
使用 vRealize Orchestrator 系統儀表板	59
在 vRealize Orchestrator 用戶端中使用工作流程 Token 重新執行	60
驗證 vRealize Orchestrator 工作流程	61
在 vRealize Orchestrator 用戶端中驗證工作流程並修正驗證錯誤	61
在 vRealize Orchestrator 用戶端中偵錯工作流程指令碼	62

依架構元素偵錯工作流程	63
為 Python 套件設定 Photon OS 容器	64

# 使用 VMware vRealize Orchestrator 用戶端

# 1

《使用 VMware vRealize Orchestrator 用戶端》提供 vRealize Orchestrator Client 的工作流程自動化特性及功能的相關資訊。

## 適合對象

此資訊適用於想要尋找工具來協助他們執行和管理 vRealize Orchestrator 工作流程的資深系統管理員。

# vRealize Orchestrator 用戶端簡介

# 2

使用 vRealize Orchestrator Client 管理 vRealize Orchestrator 服務和物件。

您可以在 [https://your\\_orchestrator\\_FQDN/orchestration-ui](https://your_orchestrator_FQDN/orchestration-ui) 上存取 vRealize Orchestrator Client。

UI 元素	說明
儀表板	若要收集 vRealize Orchestrator 環境和工作流程相關的有用度量資料，請使用 vRealize Orchestrator Client 儀表板和剖析功能。
工作流程	建立、編輯、排程、執行和刪除工作流程。
動作	建立、編輯和刪除動作。動作編輯器支援自動完成 vRealize Orchestrator API Explorer 中包含的常見指令碼元素。
原則	建立、編輯、執行和刪除原則。
套件	建立、刪除、匯出和匯入包含 vRealize Orchestrator 物件的套件。
組態	建立、執行和刪除組態元素。
資源	匯出、匯入和更新資源元素。
群組	具有管理員權限的使用者可以將角色指派給 vRealize Orchestrator Client 中的使用者，並將其新增至群組。
稽核記錄	檢視 vRealize Orchestrator Client 中記錄的不同事件，例如建立物件時。
Git 存放庫	建立與 Git 存放庫的整合，並使用整合來管理跨多個部署的工作流程和其他 vRealize Orchestrator 物件的開發。 請參閱 <a href="#">如何使用 Git 分支管理 vRealize Orchestrator 物件詳細目錄</a> 。
已刪除的項目	還原已刪除的 vRealize Orchestrator Client 物件，例如工作流程、動作、原則、組態元素和資源元素。
API Explorer	探索 vRealize Orchestrator Client 中可用的 API 命令。 <b>備註</b> vRealize Orchestrator Client 會透過 REST Proxy 與 vRealize Orchestrator REST API 進行通訊。

本章節討論下列主題：

- [vRealize Orchestrator 用戶端使用量儀表板](#)
- [vRealize Orchestrator Client 中的內容組織整理](#)

## vRealize Orchestrator 用戶端使用量儀表板

vRealize Orchestrator Client 儀表板提供可用於監控、管理和疑難排解 vRealize Orchestrator Client 工作流程的實用工具。

vRealize Orchestrator Client 儀表板上的資訊分散在五個面板中。

Windows	說明
工作流程執行	提供有關執行中、等待中和已失敗工作流程執行數量的視覺資料。
常用工作流程	顯示新增至常用列表的工作流程。
等待輸入	顯示需要進一步使用者互動的擱置中工作流程執行。這些工作流程也會顯示在使用者介面右上角的通知功能表中。
最近的工作流程執行	管理最近的工作流程執行。顯示工作流程執行的名稱、狀態、開始日期和結束日期。
需要注意	顯示失敗的工作流程執行和工作流程執行效能度量。

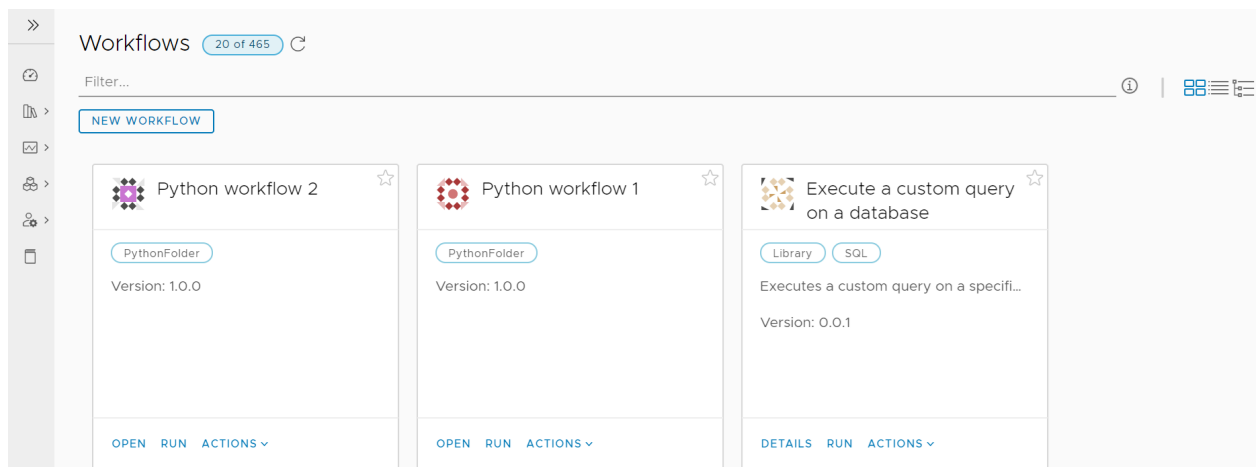
## vRealize Orchestrator Client 中的內容組織整理

管理 vRealize Orchestrator 物件詳細目錄在 vRealize Orchestrator Client 中的顯示方式。

vRealize Orchestrator Client 針對工作流程、動作、原則、資源和組態等物件支援三種不同的視圖類型：卡視圖、清單視圖和樹狀結構視圖。您可以從頁面的右上角變更目前的視圖類型。

### 卡視圖

卡視圖是 vRealize Orchestrator Client 中使用的預設視圖類型。個別詳細目錄物件 (例如工作流程) 的相關資訊會顯示在單獨的卡元素中。





## 清單視圖

清單視圖會以清單組織整理的形式顯示 vRealize Orchestrator 物件的相關資訊。如需有關可對物件執行的動作的詳細資訊，請按一下物件左邊的垂直省略符號圖示。

Name	Tags	Version	Description
Python workflow 2	PythonFolder	1.0.0	
Python workflow 1	PythonFolder	1.0.0	
Execute a custom query on a database	Library SQL	0.0.1	Executes a custom query on a specified database and returns the number of affected rows. You can run the workflow to update, delete, and insert queries.
JDBC URL generator	Library JDBC	0.0.8	Solicits information to generate a connection URL for JDBC database connections. The workflow emits the connection string it generates as output via the system log, and confirms the string can create a connection to the specified database.

## 樹狀結構視圖

您可以在樹狀結構檢視中的階層式資料夾下組織整理物件詳細目錄。每個 vRealize Orchestrator 物件類型都有一個根層級資料夾。您無法在根資料夾下建立新物件，例如工作流程。必須建立在根資料夾下組織整理的單獨資料夾。每個資料夾包含的工具可協助您管理其內容，例如內容篩選器。

**備註** 每個資料夾都有單獨的內容篩選器。無法跨資料夾篩選內容。

如需有關資料夾的詳細資訊，請參閱在 [vRealize Orchestrator 用戶端中建立資料夾或子資料夾](#)。

**備註** 當您從樹狀結構視圖中選取物件時，將以唯讀模式開啟。若要編輯物件內容，例如工作流程變數或工作流程架構，請按一下頂部選項功能表中的**編輯**。

Name	Type
Python workflow 1	Workflow
Python workflow 2	Workflow

## 在 vRealize Orchestrator 用戶端中建立資料夾或子資料夾


使用階層式資料夾結構組織整理 vRealize Orchestrator 物件。

您可以建立資料夾和子資料夾，以組織整理下列類型的 vRealize Orchestrator 物件：

- 工作流程
- 動作

- 原則
- 組態元素
- 資源元素

#### 程序


- 1 登入 vRealize Orchestrator Client。
- 2 從左側導覽窗格中，選取物件頁面，例如**工作流程**。
- 3 從右上方選取樹狀結構視圖圖示 ()。
- 4 (選擇性) 若要建立子資料夾，請從左側的樹狀結構視圖中選取父系資料夾。
- 5 按一下**新增資料夾**。
- 6 輸入名稱和說明，然後按一下**儲存**。
- 7 將物件或子資料夾新增至新建立的資料夾。
- 8 (選擇性) 若要編輯資料夾名稱，請選取**動作 > 編輯**。

## 在 vRealize Orchestrator Client 中移動物件和資料夾

將內容移至其他資料夾以重新組織整理 vRealize Orchestrator 內容。

您無法在動作模組之間移動動作，也無法將任何物件移至根資料夾。根資料夾包括主要物件資料夾和子資料夾，但無法用於儲存物件。

#### 程序

- 1 登入 vRealize Orchestrator Client。
- 2 從左側導覽窗格中，選取物件頁面，例如**工作流程**。
- 3 從右上方選取樹狀結構視圖圖示 ()。
- 4 展開樹狀結構視圖，然後選取您要移動的物件或資料夾。
- 5 將物件或資料夾拖曳至新的父系資料夾。

---

**備註** 您也可以直接從物件編輯器將物件移到新資料夾。在**摘要**索引標籤上，按一下**選取資料夾**，然後為物件選取新的父系資料夾。另一個移動選項是從資料夾頁面上的資料表中選取物件。此選項對於執行包含多個 vRealize Orchestrator 物件的批次移動作業非常有用。

---


## 在 vRealize Orchestrator Client 中刪除資料夾或子資料夾

從 vRealize Orchestrator Client 中刪除過時的資料夾或子資料夾。

無法刪除每個 vRealize Orchestrator 物件類型對應的根層級資料夾。

#### 程序

- 1 登入 vRealize Orchestrator Client。

- 2 從左側導覽窗格中，選取物件頁面，例如**工作流程**。
- 3 從右上方選取樹狀結構視圖圖示 ()。
- 4 勾選您要刪除之資料夾旁邊的核取方塊。

---

**備註** 若要刪除子資料夾，請從樹狀結構視圖中選取父系資料夾，然後勾選此核取方塊。

---

- 5 按一下**刪除**。
- 6 如果選取的資料夾是空白的。
  - a 請確認您要刪除資料夾。
  - b 按一下**刪除**。
- 7 如果選取的資料夾包含 vRealize Orchestrator Client 物件或子資料夾。
  - a 請確認您要刪除資料夾。
  - b 按一下**刪除**。

您會收到訊息無法刪除項目「your\_folder\_name」：資料夾「your\_folder\_name」不是空的。

- c 若要刪除資料夾及其所有內容，請按一下**強制刪除**。
- d 確認您要刪除資料夾，然後按一下**刪除**。

---

**備註** 還可以從資料夾功能表中所包含的資料表選取多個物件，以執行批次刪除。

---

# 設定 vRealize Orchestrator Client

# 3

若要充分利用 vRealize Orchestrator Client 的功能，您必須設定使用者權限並瞭解如何使用版本歷程記錄來管理物件。

本章節討論下列主題：

- [vRealize Orchestrator 角色和群組](#)
- [vRealize Orchestrator 物件的版本歷程記錄](#)

## vRealize Orchestrator 角色和群組

vRealize Orchestrator 管理員可以設定權限，以控制 vRealize Orchestrator Client 中功能和內容的存取權。存取權限分為使用者角色和群組權限。

角色可控制使用者能夠檢視和使用哪些 vRealize Orchestrator Client 功能。對角色管理功能的存取權取決於 vRealize Orchestrator 環境的授權類型。

表 3-1. 對 vRealize Orchestrator 角色管理的授權型存取

授權	驗證	
	vSphere	vRealize Automation
<b>vSphere</b>	不支援角色管理。群組僅支援「執行」權限。	
<b>vRealize Automation</b>	在 vRealize Orchestrator 用戶端中管理角色。 請參閱在 <a href="#">vRealize Orchestrator Client</a> 中指派角色。	透過 vRealize Automation 中的身分識別和存取管理來管理角色。 請參閱在 <a href="#">vRealize Automation</a> 中設定 vRealize Orchestrator 用戶端角色。

群組權限可控制使用者能夠檢視及使用哪些 vRealize Orchestrator Client 內容，例如工作流程、動作、原則、組態元素和資源元素。在所有使用者之間可以共用對預先設定的系統 vRealize Orchestrator 內容 (例如，標準工作流程和動作) 的存取權，除非透過群組權限設定了存取權。

對於具有管理員和檢視者角色的使用者，其存取權限不受群組權限的限制。對於未獲指派角色的使用者或具有工作流程設計人員角色的使用者，存取權限會取決於其獲指派的群組。您可以修改這些使用者的群組權限來延伸其存取權限。透過這種方式，您可將使用者組織整理到一般專案中。例如，您可以建立一個包含負責開發自訂 vRealize Orchestrator 外掛程式使用者的群組，並僅允許他們修改其群組專屬的內容。

表 3-2. vRealize Orchestrator 使用者角色和群組權限

角色	存取權限
管理員	<p>管理員可以存取所有 vRealize Orchestrator 用戶端功能和內容，包括由特定群組建立的內容。負責設定使用者角色、建立和刪除群組，以及將使用者新增到群組。管理員不受群組權限的限制。</p> <p><b>備註</b> 依預設，vRealize Automation 環境中用來驗證 vRealize Orchestrator 的承租人管理員，會具有<b>管理員</b>權限。</p>
檢視者	<p>檢視者具有 vRealize Orchestrator 用戶端中所有內容的唯讀存取權，但無法建立、編輯、執行或匯出內容。檢視者也可以查看所有的群組和群組內容。檢視者不受群組權限的限制。</p>
群組權限	
	<div>沒有指派的群組</div> <div>執行</div> <div>執行和編輯</div>
工作流程設計人員	<div> <ul style="list-style-type: none"> <li>■ 檢視系統內容。</li> <li>■ 檢視及執行自己的執行。</li> <li>■ 建立、執行、編輯和刪除自己的內容。</li> </ul> </div> <div> <ul style="list-style-type: none"> <li>■ 檢視系統內容</li> <li>■ 檢視及執行自己的執行。</li> <li>■ 建立、執行、編輯和刪除自己的內容。</li> <li>■ 將自己的內容新增至群組。</li> <li>■ 執行群組內容，但無法加以編輯。</li> </ul> </div> <div> <ul style="list-style-type: none"> <li>■ 檢視系統內容。</li> <li>■ 檢視及執行自己的執行。</li> <li>■ 建立、執行、編輯和刪除自己的內容。</li> <li>■ 將自己的內容新增至群組。</li> <li>■ 執行及編輯群組內容。</li> </ul> <p><b>備註</b> 不適用於使用 vSphere 進行驗證的 vRealize Orchestrator 執行個體。</p> </div>
未獲指派角色的使用者	<div> <ul style="list-style-type: none"> <li>■ 檢視自己的執行。</li> </ul> </div> <div> <ul style="list-style-type: none"> <li>■ 檢視及執行自己的執行。</li> <li>■ 檢視及執行群組內容。</li> </ul> </div> <div> <ul style="list-style-type: none"> <li>■ 檢視及執行自己的執行。</li> <li>■ 檢視及執行群組內容。</li> </ul> <p><b>備註</b> 若要能夠建立、編輯和新增內容，此群組中的使用者必須獲指派工作流程設計人員角色。</p> <p><b>備註</b> 不適用於使用 vSphere 進行驗證的 vRealize Orchestrator 執行個體。</p> </div>

## 在 vRealize Orchestrator Client 中指派角色

做為管理員，您可以將使用者新增至 vRealize Orchestrator Client 並設定可供其檢視和使用的功能。

角色管理會控制 vRealize Orchestrator 身分識別提供者中的使用者對 vRealize Orchestrator Client 功能的存取。角色管理涵蓋 vRealize Orchestrator Client 使用者介面和 API 功能。

**備註** 用戶端角色管理僅適用於使用 vRealize Automation 授權，透過 vSphere 進行驗證的 vRealize Orchestrator 執行個體。如需將角色指派給使用 vRealize Automation 進行驗證的 vRealize Orchestrator 的相關資訊，請參閱在 [vRealize Automation 中設定 vRealize Orchestrator 用戶端角色](#)。

### 程序

- 1 以管理員身分登入 vRealize Orchestrator 用戶端。

- 2 導覽至**管理 > 角色管理**。
- 3 按一下**新增**。
- 4 搜尋您想要新增至 vRealize Orchestrator Client 的使用者或群組。
- 5 選取使用者的角色。如需有關角色的詳細資訊，請參閱 [vRealize Orchestrator 角色和群組](#)。
- 6 按一下**儲存**。

## 在 vRealize Automation 中設定 vRealize Orchestrator 用戶端角色

您可以在 vRealize Automation 的**身分識別與存取管理**頁面中為 vRealize Orchestrator Client 指派服務角色。可以為使用 vRealize Automation 進行驗證的內嵌式 vRealize Orchestrator Client 和獨立 vRealize Orchestrator 執行個體指派服務角色。

vRealize Orchestrator 服務角色可管理內嵌式 vRealize Orchestrator Client 使用者可存取的功能。如需有關 vRealize Orchestrator 角色的詳細資訊，請參閱 [vRealize Orchestrator 角色和群組](#)。

---

**備註** 使用 vRealize Automation 授權，透過 vSphere 進行驗證的獨立 vRealize Orchestrator 執行個體可以直接在 vRealize Orchestrator Client 中指派角色。請參閱在 [vRealize Orchestrator Client 中指派角色](#)。

---

### 必要條件

- 確認從有效的 vIDM 執行個體匯入適當的使用者和群組。
- 在將 vRealize Orchestrator 服務角色指派給使用者之前，請確認您的使用者在 vRealize Automation 中具有指派的組織角色。請參閱《管理 vRealize Automation》中的〈在 vRealize Automation 中管理使用者和群組〉。

### 程序

- 1 從右上方的標頭下拉式功能表中，選取**身分識別與存取管理**選項。
- 2 在**作用中使用者**索引標籤上，搜尋您要指派給 vRealize Orchestrator 的使用者的電子郵件地址。
- 3 選取使用者旁邊的核取方塊，然後按一下**編輯角色**。
- 4 按一下**新增服務存取權**。
- 5 從左側下拉式功能表中選取 **Orchestrator**。
- 6 從右側下拉式功能表中，選取您要指派給使用者的角色。
- 7 按一下**儲存**。

## 在 vRealize Orchestrator Client 中建立群組

做為管理員，您可以使用群組來設定使用者可以在 vRealize Orchestrator Client 中檢視和存取的 vRealize Orchestrator 內容。

您可以使用 vRealize Orchestrator Client 將群組權限設為 vRealize Orchestrator 工作流程、動作、原則、組態元素、資源元素和套件。

---

**備註** 對於使用 vSphere 進行驗證的 vRealize Orchestrator 執行個體的使用者，只能擁有**執行**群組權限。

---

程序

- 1 以管理員身分登入 vRealize Orchestrator 用戶端。
- 2 導覽至**管理 > 群組**。
- 3 按一下**新增群組**。
- 4 在**摘要**索引標籤上，新增群組的名稱與說明。
- 5 在**使用者**索引標籤上，按一下**新增**。
  - a 搜尋您想要新增至群組的使用者。
  - b 指派群組權限給使用者。
  - c 按一下**新增**。
- 6 在**項目**索引標籤上，將 vRealize Orchestrator 物件新增至群組。

---

**備註** 您也可以在建 vRealize Orchestrator Client 中建立物件時，將該物件新增至現有群組。若要新增物件，請從物件編輯器的**摘要/一般**索引標籤上的**可存取者**下拉式功能表中選取群組。

---

- 7 按一下**儲存**。

## vRealize Orchestrator 物件的版本歷程記錄

vRealize Orchestrator Client 會保留每個 vRealize Orchestrator 物件的版本歷程記錄。使用版本歷程記錄，您可以比較不同的 vRealize Orchestrator 物件版本以及還原到先前的版本。

當您儲存物件時，vRealize Orchestrator 會建立每個 vRealize Orchestrator 物件的版本歷程記錄。

vRealize Orchestrator 物件的後續變更會建立新的版本歷程記錄。先前的版本歷程記錄會保留，並且可用於追蹤物件的變更以及將物件還原到先前的版本。將物件還原到先前的版本會建立新的版本歷程記錄。

vRealize Orchestrator Client 可追蹤下列 vRealize Orchestrator 物件的版本歷程記錄：

- 工作流程
- 動作
- 套件
- 原則
- 資源元素

## ■ 組態元素

**備註** 產生的工作流程不會顯示在工作流程版本歷程記錄中。例如，**為資料表產生 CRUD 工作流程** 工作流程所產生的工作流程不會顯示在**版本歷程記錄**索引標籤上，且無法推送至任何已設定的 Git 存放庫。若要在 vRealize Orchestrator 版本歷程記錄中納入這些工作流程，請複製產生的工作流程。

您可以從物件編輯器頁面的**版本歷程記錄**索引標籤存取物件的版本歷程記錄。如果您嘗試與另一個使用者同時編輯物件，可能會發生合併衝突。若要解決合併衝突，請按一下錯誤訊息右側的**解決**。**解決衝突**視窗中有三個選項：

- **使用其他使用者所做的變更。**使用其他使用者所做的變更以解決合併衝突。
- **使用我們所做的變更。**使用您的變更以解決合併衝突。
- **解決。**透過編輯顯示的變更模型解決合併衝突。如果提供的模型無效，則無法使用此選項。

## 將工作流程還原至先前版本

您可以將工作流程還原到先前儲存的版本。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 工作流程**，然後選取工作流程。
- 3 選取**版本歷程記錄**索引標籤。
- 4 若要檢視版本之間的比較，請選取工作流程版本，然後從**差異比較**下拉式功能表中選取其他版本。  
視窗顯示目前工作流程版本與選取工作流程版本間的差異。
- 5 若要將工作流程還原到其他版本，請按一下**還原**。  
工作流程狀態還原為選取版本的狀態。

**備註** 您也可以從圖形差異工具視圖還原工作流程版本。請參閱**工作流程版本之間的視覺比較**。

## 工作流程版本之間的視覺比較

使用圖形差異工具比較工作流程版本之間的變更。

依預設，vRealize Orchestrator 版本歷程記錄會在 YAML 表單中顯示工作流程版本之間的差異。您還可以在不同的工作流程版本之間執行視覺比較。您可以檢視以下內容的變更：

- 一般工作流程資訊，例如版本號碼和工作流程說明。
- 工作流程中使用的變數。
- 工作流程的輸入和輸出參數。
- 工作流程架構。

必要條件

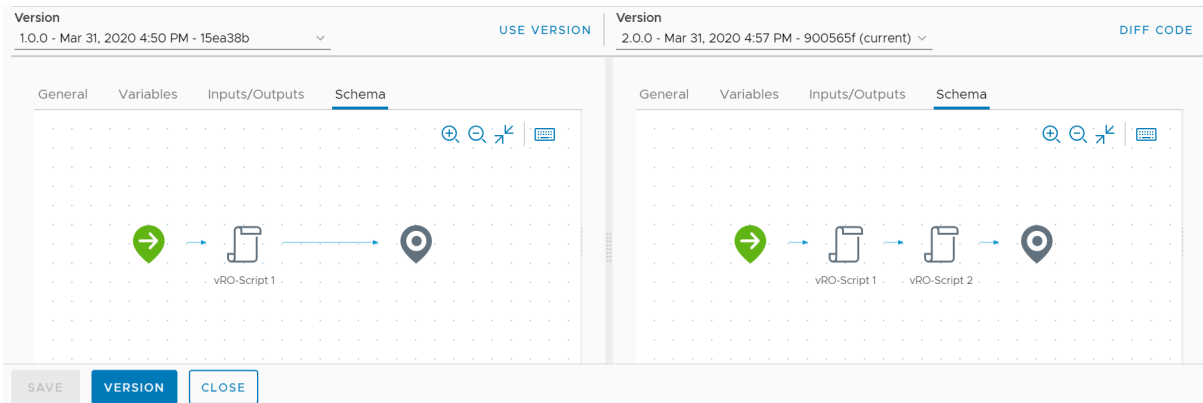
建立工作流程。



## 程序

- 1 登入 vRealize Orchestrator Client。
- 2 導覽至**程式庫 > 工作流程**，然後選取其中一個工作流程。
- 3 編輯工作流程的內容。  
例如，您可以在**架構**索引標籤上新增額外的**可編寫指令碼**的工作元素。
- 4 按一下**儲存**。
- 5 選取**版本歷程記錄**索引標籤。
- 6 從右上方選取**直觀比較差異**。

您現在可以在兩個選取的工作流程版本之間執行視覺比較。可以從**版本**下拉式功能表中選取要比較的版本。



- 7 (選擇性) 您可以透過選取**使用版本**將工作流程還原至其他版本。

## 使用 Git 將 vRealize Orchestrator 內容詳細目錄重設為先前的狀態

透過使用先前的 Git commit，您可以將 vRealize Orchestrator 內容重設為先前狀態。

您可以透過選取特定的 commit，將 vRealize Orchestrator 內容重設為先前的狀態。

### 必要條件

- 設定與 GitHub 或 GitLab 存放庫的連線。請參閱**設定與 Git 存放庫的連線**。
- 將本機變更集推送到已設定的 Git 存放庫。

## 程序

- 1 登入 vRealize Orchestrator Client。
- 2 導覽至**管理 > Git 歷程記錄**。
- 3 選取要重設到的變更集，然後按一下**重設為此**。

**4** 確認您要重設為此特定 commit，然後按一下**確定**。

vRealize Orchestrator 內容詳細目錄會重設為此 commit 中指定的狀態。相關的 vRealize Orchestrator 內容會還原到先前的版本。如果在推送 commit 時此內容不存在，則會將其從詳細目錄中移除。

**後續步驟**

若要將 vRealize Orchestrator 詳細目錄還原為儲存至 Git 存放庫的最新狀態，請從 **Git 歷程記錄** 視窗執行 Pull 命令。

# vRealize Orchestrator 使用案例

# 4

這些使用案例示範了 vRealize Orchestrator 平台的部分功能。

這些使用案例僅提供範例值。您自己的環境結構和命名慣例會有所不同。

本章節討論下列主題：

- 如何使用 Python 在 vRealize Orchestrator 中整合 Amazon Web Services
- 如何使用 Git 分支管理 vRealize Orchestrator 物件詳細目錄
- 如何使用第三方模組呼叫 vRealize Automation 專案 API

## 如何使用 Python 在 vRealize Orchestrator 中整合 Amazon Web Services

此 vRealize Orchestrator 使用案例顯示如何使用 Python 擴充 vRealize Orchestrator 部署功能的範例。

您可以在動作和工作流程指令碼中使用下列執行階段：

- Python 3.7
- Node.js 14
- PowerCLI 11/Powershell 6.2
- PowerCLI 12.3.0/Powershell 7.1

---

**備註** PowerCLI 執行階段包括 PowerShell 和下列模組：VMware.PowerCLI、PowerNSX、PowervRA。

---

**重要** 僅當您的 vRealize Orchestrator 部署使用 vRealize Automation 授權時，才能使用新的執行階段。

---

此使用案例示範了如何建立可呼叫 Amazon Web Services (AWS) 中 EC2 執行個體的 Python 指令碼。

**重要** 在您開始開發自訂指令碼之前，請先確認您熟悉在 vRealize Orchestrator 中使用 Python、Node.js 和 PowerShell 指令碼的核心概念。請參閱 [Python](#)、[Node.js](#) 和 [PowerShell](#) 指令碼的核心概念。

## 程序

### 1 建立初始 Python 指令碼

在本機上，建立 Python 指令碼，並將指令碼和 boto3 程式庫封裝為 ZIP 資料夾。

### 2 建立 Amazon Web Services 動作

建立使用 Python 指令碼的 vRealize Orchestrator 動作。

### 3 偵錯 Amazon Web Services 動作

原始版本的 Python 指令碼刻意含有內建錯誤，以便您可以瞭解如何偵錯指令碼。

### 4 更新 Amazon Web Services 動作

匯入更新的 Python 指令碼，然後再次執行動作。

## 建立初始 Python 指令碼

在本機上，建立 Python 指令碼，並將指令碼和 boto3 程式庫封裝為 ZIP 資料夾。

### 必要條件

- 下載並安裝 Python 3。請參閱 [Python 下載頁面](#)。
- 下載並安裝 Visual Studio Code。請參閱 [Visual Studio Code 下載頁面](#)。
- 請確認是否已安裝適用於 Visual Studio Code 的 Python 延伸。請參閱 [Visual Studio Marketplace](#)。

## 程序

### 1 在本機上，建立 vro-python-aws 資料夾，然後在其上安裝 boto3 Python SDK。

```
mkdir vro-python-aws
cd vro-python-aws
mkdir lib
pip install boto3 -t lib/
```

### 2 開啟編輯器，然後建立主要 Python 指令碼。在此使用案例中，您使用的是 Visual Studio Code。

```
import boto3

def handler(context, inputs):
    ec2 = boto3.resource('ec2')
    filters = [{
        'Name': 'instance-state-name',
        'Values': ['running']
    }]
```

```
instances = ec2.instances.filter(Filters=filters)
for instance in instances:
    print('Instance: ' + instance.id)
```

此 Python 指令碼會列出指定區域中所有正在執行的 EC2 執行個體。

- 3 將建立的指令碼在 vro-python-aws 資料夾中另存為 main.py 檔案。
- 4 登入命令列介面。
- 5 導覽至 vro-python-aws 資料夾。

```
cd vro-python-aws
```

- 6 建立包含 Python 指令碼的 ZIP 套件。

```
zip -r --exclude=*.zip -X vro-python-aws.zip .
```

---

**備註** 您也可以使用 ZIP 公用程式工具 (例如 7-Zip) 來建立 ZIP 套件。

---

## 結果

您已建立基礎 Python 指令碼，並準備好將其匯入至 vRealize Orchestrator 部署。

## 建立 Amazon Web Services 動作

建立使用 Python 指令碼的 vRealize Orchestrator 動作。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至 **程式庫 > 動作**。
- 3 按一下 **新增動作**。
- 4 在 **一般** 索引標籤上，針對動作輸入名稱、模組與版本號碼。
- 5 在 **指令碼** 索引標籤上，選取 **Python 3.7** 做為執行階段，並選取 **Zip** 做為指令碼類型。
- 6 按一下 **匯入**。
- 7 瀏覽至 vro-python-aws 資料夾，然後選取包含 Python 指令碼的 ZIP 套件。
- 8 在 **項目處理常式** 文字方塊中，輸入 **main.handler**。

---

**備註** 動作的項目處理常式是以匯入的 ZIP 套件中的主要指令碼為基礎。由於主要指令碼位於名為 main.py 的檔案以及名為 **handler** 的函數中，因此項目處理常式必須為 **main.handler**。如果您的主要指令碼檔案有不同的標題，請相應地變更項目處理常式值。

---

- 9 儲存動作，然後按一下 **執行**。

動作執行將遇到錯誤。

## 10 選取記錄索引標籤。

動作執行的記錄顯示「botocore.exceptions.NoRegionError: You must specify a region.」錯誤訊息。這是預期行為，因為初始 Python 指令碼未定義區域。

後續步驟

偵錯 Python 指令碼。請參閱偵錯 [Amazon Web Services](#) 動作。

## 偵錯 Amazon Web Services 動作

原始版本的 Python 指令碼刻意含有內建錯誤，以便您可以瞭解如何偵錯指令碼。

必要條件

登入您的 Amazon Web Services (AWS) 帳戶，並建立專門用於此使用案例的 IAM 使用者。請參閱〈[在 AWS 帳戶中建立 IAM 使用者](#)〉。IAM 使用者必須具備下列權限：

```
"Effect": "Allow",
"Action": "ec2:DescribeInstances",
"Resource": "*"

```

程序

### 1 準備 vRealize Orchestrator Appliance。

**注意** 請勿在生產 vRealize Orchestrator 部署中偵錯指令碼。從用於開發和測試的單一節點 vRealize Orchestrator 部署進行偵錯。

- a 以 **root** 身分透過 SSH 登入 vRealize Orchestrator Appliance 命令列。
- b 執行 `vracli dev tools` 命令。
- c 系統會提示您確認是否要繼續。輸入 **yes** 繼續，或輸入 **no** 取消。

**重要** 透過執行 `vracli dev tools` 命令，可以開啟偵錯 Python 指令碼所需的連接埠。在偵錯程序期間，必須保持開啟目前的 SSH 工作階段。

### 2 啟動偵錯組態。

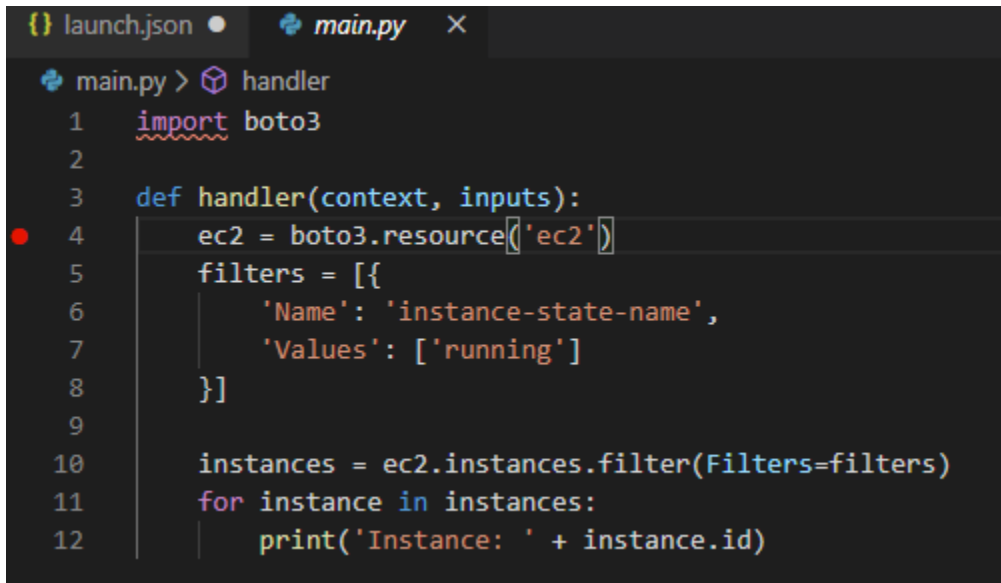
- a 登入 vRealize Orchestrator 用戶端。
- b 開啟 AWS 動作，然後按一下**偵錯**。  
偵錯程序隨即開始，且動作執行已暫停。
- c 選取**偵錯組態**索引標籤。  
此索引標籤包含一個 .json 組態，您可以從遠端將其連結至 IDE 來偵錯 Python 指令碼。
- d 手動複製組態內容，或按一下**複製到剪貼簿**。

### 3 偵錯 Python 指令碼。

- a 開啟 Visual Studio Code。
- b 開啟 vro-python-aws 資料夾。
- c 從頂部導覽窗格中，選取**執行 > 開啟組態**。
- d 選取 **Python 檔案**。
- e 將 "version" 和 "configuration" 屬性保留在目前位置中，並貼上從 vRealize Orchestrator 用戶端複製的 .json 組態的內容。產生的 launch.json 檔案必須類似如下：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "request": "attach",
      "port": 18281,
      "name": "vRO Python debug 8302f4c7-5beb-40da-848a-5003c0296f7b",
      "host": "es-sof-vc-vm-225-190.sof-mbu.eng.vmware.com",
      "type": "python",
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}",
          "remoteRoot": "/var/run/vco-polyglot/function"
        }
      ]
    }
  ]
}
```

- f 選取 main.py 指令碼檔案，然後將中斷點新增至 `ec2 = boto3.resource('ec2')` 行。



```
{ launch.json } main.py x
main.py > handler
1  import boto3
2
3  def handler(context, inputs):
4      ec2 = boto3.resource('ec2')
5      filters = [{
6          'Name': 'instance-state-name',
7          'Values': ['running']
8      }]
9
10     instances = ec2.instances.filter(Filters=filters)
11     for instance in instances:
12         print('Instance: ' + instance.id)
```

- g 從頂部導覽窗格中，選取**執行 > 開始偵錯**。

- h 當偵錯工具到達中斷點時，執行跳過作業。

偵錯執行指示 Python 指令碼缺少指定的區域和 AWS 存取金鑰。

- i 返回開啟的 vRealize Orchestrator Appliance 工作階段，然後按 **Enter** 以關閉為此偵錯工作階段開啟的連接埠。

#### 4 將遺失的資訊新增至 Python 指令碼。

- a 在 Visual Studio Code 中，建立名為 awsconfig 的檔案，其中包含 IAM 使用者的 AWS 存取金鑰，以及您想要使用 Python 指令碼執行 ping 動作的 AWS 區域。

```
[default]
aws_access_key_id=your key ID
aws_secret_access_key=your secret access key
region=your-region
```

- b 將 awsconfig 在 vro-python-aws 資料夾中另存為組態檔 (.cfg)。
- c 開啟 main.py 檔案並進行修改，使 boto3 程式庫能夠使用 awsconfig.cfg 檔案。

```
import boto3

import os
os.environ['AWS_CONFIG_FILE'] = os.getcwd() + '/awsconfig.cfg'

def handler(context, inputs):
    ec2 = boto3.resource('ec2')
    filters = [{
        'Name': 'instance-state-name',
        'Values': ['running']
    }]

    instances = ec2.instances.filter(Filters=filters)
    for instance in instances:
        print('Instance: ' + instance.id)
```

- d 建立新的 ZIP 套件，其中包含 main.py 檔案、awsconfig.cfg 檔案和 boto3 程式庫。

```
zip -r --exclude=*.zip -X vro-python-aws.zip .
```

**備註** 您也可以使用 ZIP 公用程式工具 (例如 7-Zip) 來建立 ZIP 套件。

## 更新 Amazon Web Services 動作

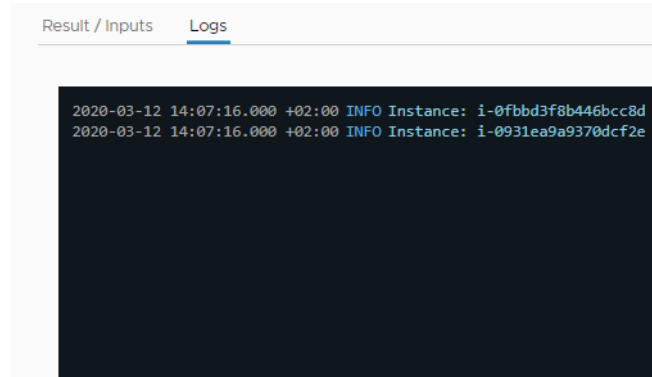
匯入更新的 Python 指令碼，然後再次執行動作。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至 **程式庫 > 動作**，然後選取原始 Amazon Web Services (AWS) 動作。



- 3 (選擇性) 在一般索引標籤上，變更版本號碼。
- 4 移除舊的 ZIP 套件，然後按一下匯入。
- 5 選取更新的 ZIP 套件。
- 6 儲存動作，然後按一下執行。
- 7 動作執行完成後，選取記錄索引標籤。



記錄會顯示此動作查詢的 EC2 執行個體。

#### 後續步驟

建立使用已更新的 AWS 動作做為**動作元素**的 vRealize Orchestrator 工作流程。

## 如何使用 Git 分支管理 vRealize Orchestrator 物件詳細目錄

使用分支來組織整理在 Git 存放庫中管理 vRealize Orchestrator 內容的方式。

透過使用 Git，您可以提供集中式存放庫，以提高 vRealize Orchestrator 開發人員的靈活性。例如，您可以使用 Git 管理跨多個 vRealize Orchestrator 環境的工作流程開發。

---

**備註** 若要使用 Git 管理物件詳細目錄，您的 vRealize Orchestrator 部署必須使用 vRealize Automation 授權。如需詳細資訊，請參閱《安裝和設定 vRealize Orchestrator》中的〈使用授權啟用 vRealize Orchestrator 功能〉。

---

您現在可以將物件推送到分支以及從分支提取物件。您可以使用分支管理特定 vRealize Orchestrator 物件群組的開發，然後再將其合併回主要分支。

在此使用案例中，您使用 GitLab 專案來管理使用 Python 執行階段的 vRealize Orchestrator 物件。此使用案例表示 vRealize Orchestrator 中的 Git 功能範例，並不表示功能範圍的限制。

---

**備註** 如果您更熟悉 GitHub，則可以針對此使用案例使用 GitHub 存放庫。

---

#### 程序

##### 1 準備 GitLab 環境

為 vRealize Orchestrator Python 物件建立 Git 分支。

##### 2 設定與 Git 存放庫的連線

做為**管理員**，您可以設定 vRealize Orchestrator 部署與 Git 存放庫或專案之間的連線。

### 3 將變更推送至 Git 存放庫

將對本機 vRealize Orchestrator 物件所做的變更推送至整合式 Git 存放庫。在此使用案例中，我們會將對基於 Python 的 vRealize Orchestrator 動作的變更推送至特定的 Git 分支。

## 準備 GitLab 環境

為 vRealize Orchestrator Python 物件建立 Git 分支。

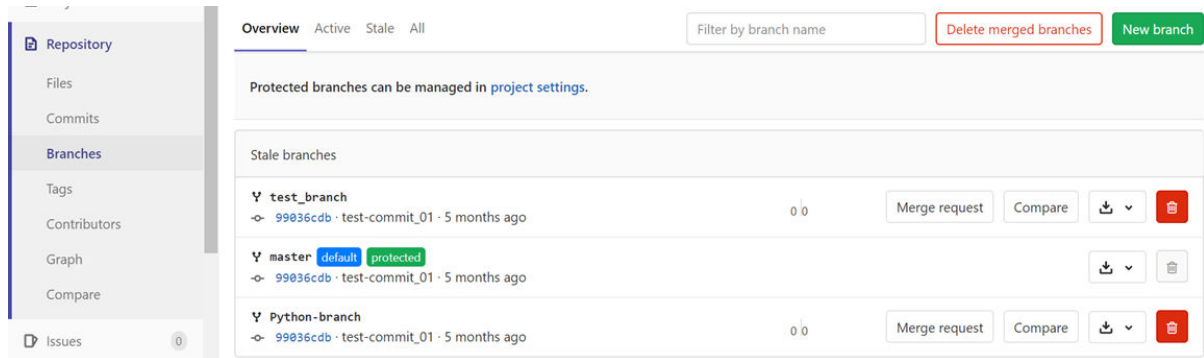
必要條件

為 vRealize Orchestrator 環境建立 GitLab 專案。請參閱〈[建立專案](#)〉。

程序

- 1 登入您的 GitLab 帳戶。
- 2 導覽至 GitLab 專案。
- 3 在左側導覽窗格中，選取**存放庫 > 分支**。
- 4 在**概觀**索引標籤上，按一下**新增分支**。
- 5 在分支名稱下，輸入 **Python-branch**。
- 6 保留**建立自**選項為**主要分支**。
- 7 按一下**建立分支**。

您已為基於 Python 的 vRealize Orchestrator 物件建立分支。



## 設定與 Git 存放庫的連線

做為**管理員**，您可以設定 vRealize Orchestrator 部署與 Git 存放庫或專案之間的連線。

若要使用 Git 管理 vRealize Orchestrator 物件詳細目錄，您必須使用 vRealize Orchestrator Client 設定與 Git 存放庫的連線。

必要條件

- 確認您的 vRealize Orchestrator 環境使用 vRealize Automation 授權。

- 為 GitLab 專案產生存取 Token，並將其複製到剪貼簿以便在設定程序期間使用。請參閱〈[建立個人存取 Token](#)〉。

---

**備註** 在此使用案例中，您使用的是 GitLab 專案。如果您更熟悉 GitHub，可以使用 GitHub 存放庫。如需產生 GitHub Token 的相關資訊，請參閱〈[為命令列建立個人存取 Token](#)〉。

---

#### 程序

- 1 以**管理員**身分登入 vRealize Orchestrator Client。
- 2 導覽至**管理 > Git 存放庫**。
- 3 按一下**新增存放庫**。
- 4 輸入 Git 存放庫的 URL 位址。

例如，<https://gitlab.com/myusername/my-vro-repo>。

---

**備註** 也可以使用 SSH 通訊協定建立連線。

---

- 5 輸入 Git 設定檔的使用者名稱。
- 6 輸入 Git 存放庫的存取 Token。
- 7 若要驗證與 Git 存放庫的連線，請按一下**驗證**。
- 8 (選擇性) 變更用於在 vRealize Orchestrator Client 中識別存放庫的名稱。
- 9 (選擇性) 為已連線的 Git 存放庫新增簡短說明。
- 10 若要啟動已連線的 Git 存放庫，請按一下**建立作用中存放庫**。

---

**備註** 一次只能有一個 Git 存放庫處於作用中狀態。您可以從 **Git 存放庫** 頁面變更作用中 Git 存放庫。

---

- 11 選取要將變更推送到的分支。在此使用案例中，您使用的是 **Python-branch**。請參閱[準備 GitLab 環境](#)。

---

**備註** 完成初始 Git 設定後，您可以隨時變更所選的 Git 分支。

---

- 12 若要完成設定程序，請按一下**儲存**。

#### 後續步驟

導覽回到 **Git 存放庫** 功能表，並確認存放庫狀態為**作用中**。

## 將變更推送至 Git 存放庫

將對本機 vRealize Orchestrator 物件所做的變更推送至整合式 Git 存放庫。在此使用案例中，我們會將對基於 Python 的 vRealize Orchestrator 動作的變更推送至特定的 Git 分支。

您可以將本機變更集推送至 Git 存放庫。每個變更集可包含一或多個已修改的 vRealize Orchestrator 物件。

---

**備註** 將變更集推送和捨棄至 Git 存放庫的程序不受群組權限的限制。因此，一個群組中的工作流程開發人員可以推送或捨棄其他開發人員所做的本機變更。

---

#### 必要條件

- 確認您已建立 Git 分支。請參閱[準備 GitLab 環境](#)。
- 確認您已設定與 Git 存放庫的連線。請參閱[設定與 Git 存放庫的連線](#)。
- 確認您的 Git 整合已設定為將變更推送至 **Python-branch** Git 分支。
- 建立基於 Python 的 vRealize Orchestrator 物件。例如，請參閱[如何使用 Python 在 vRealize Orchestrator 中整合 Amazon Web Services](#)。

#### 程序

- 1 登入 vRealize Orchestrator Client。
- 2 編輯 Python 動作。
  - a 導覽至**程式庫 > 動作**，然後選取您的 Python 動作。
  - b 對動作進行一些細微的變更，例如變更說明。
  - c 儲存動作。

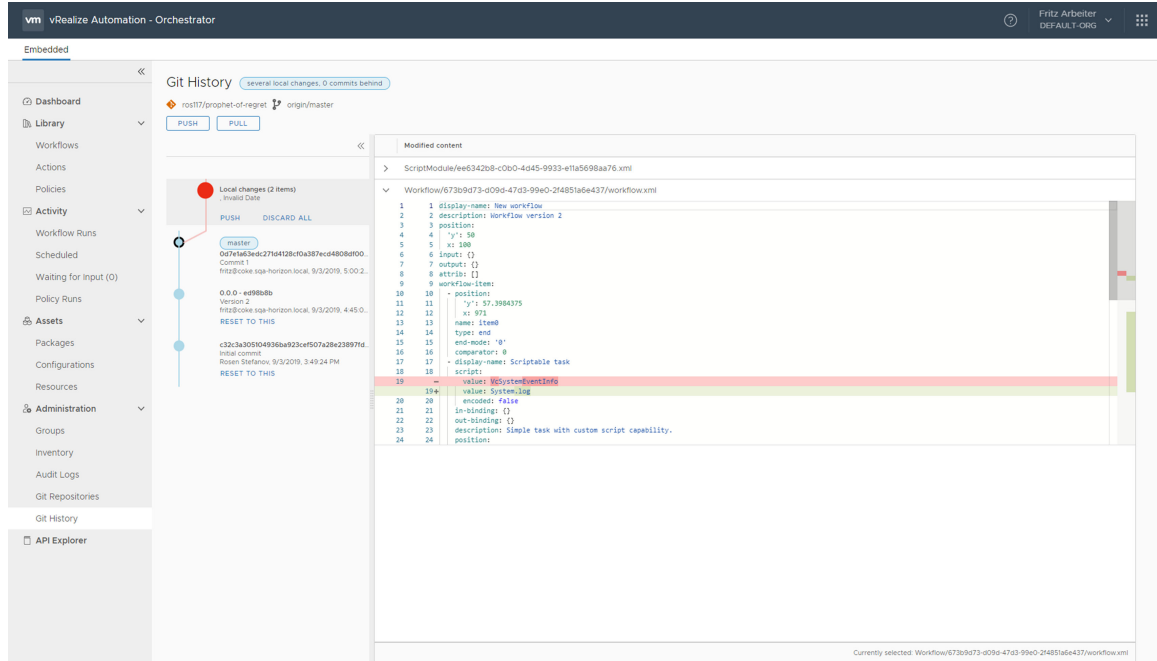
### 3 將變更推送至 Git 存放庫。

**備註** 您也可以透過按一下物件編輯器底部顯示的**版本**選項，在每個物件層級推送本機變更。

#### a 導覽至**管理 > Git 歷程記錄**。

**Git 歷程記錄**會顯示本機版本分支與所選 Git 存放庫分支之間的目前差異。您可以展開任何已修改的 vRealize Orchestrator 物件的項目，以檢視版本差異。

**備註** 可以透過選取**全部捨棄**捨棄本機變更集。



#### b 按一下**推送**。

#### c 輸入 commit 標題。

#### d (選擇性) 輸入 commit 的簡短說明。

#### e 選取要推送至 Git 存放庫的 Python 動作變更。

### 4 若要完成將本機變更集推送至 Git 存放庫，請按一下**推送**。

後續步驟

在 Git 分支中完成開發後，請將其與主要分支合併。請參閱〈[如何建立合併要求](#)〉。

## 如何使用第三方模組呼叫 vRealize Automation 專案 API

此 vRealize Orchestrator 使用案例說明如何使用第三方模組呼叫 vRealize Automation 專案 API。

您可以在動作和工作流程指令碼中使用下列執行階段：

#### ■ Python 3.7

- Node.js 14
- PowerCLI 11/Powershell 6.2
- PowerCLI 12.3.0/Powershell 7.1

---

**備註** PowerCLI 執行階段包括 PowerShell 和下列模組：VMware.PowerCLI、PowerNSX、PowervRA。

---

在此使用案例中，您將瞭解如何建立使用第三方相依性模組連線至 vRealize Automation 專案 API 的 vRealize Orchestrator 動作。

---

**重要** 在您開始開發自訂指令碼之前，請先確認您熟悉在 vRealize Orchestrator 中使用 Python、Node.js 和 PowerShell 指令碼的核心概念。請參閱 [Python](#)、[Node.js](#) 和 [PowerShell](#) 指令碼的核心理念。

---

## 建立呼叫 vRealize Automation 專案 API 的 Python 指令碼

建立使用 Python 呼叫 vRealize Automation 專案 API 的範例指令碼。

### 必要條件

確認您已安裝 Python 3 和 PIP 套件安裝程式。請參閱 [〈Python 下載頁面〉](#) 和 [〈Python 套件索引〉](#)。

### 程序

- 1 在本機電腦上，開啟命令列 Shell。
- 2 建立 vro-python-vra 資料夾。

```
mkdir vro-python-vra
```

- 3 導覽至 vro-python-vra 資料夾。

```
cd vro-python-vra
```

- 4 建立名為 handler.py 的 Python 指令碼。

```
touch handler.py
```

handler.py 指令碼必須定義一個接受兩個引數、vRealize Orchestrator 工作流程執行的內容和繫結的 vRealize Orchestrator 輸入的函數。

```
def handler(context, inputs):
    print('Hello, your inputs were ' + inputs)
    return None
```

**備註** 透過使用標準記錄程式庫，您在使用指令碼的動作中記錄的所有內容也會顯示在工作流程記錄中。必須在 vRealize Orchestrator 用戶端中為指令碼的輸入和傳回設定對應的輸入參數和傳回類型。例如，指令碼中的 vRAUrl 輸入必須在 vRealize Orchestrator 用戶端中具有名為 vRAUrl 的對應輸入參數。同樣地，如果您的指令碼傳回字串值，則 vRealize Orchestrator 用戶端中設定的傳回類型也必須是字串類型。如果動作傳回複雜物件，您可以使用 Properties 或 Composite Type 傳回類型。

## 5 安裝 Python 要求模組。

**重要** 第三方相依性模組必須安裝在主要 vro-python-vra 指令碼資料夾的根層級資料夾中。在此使用案例中，您可以為要求模組建立 lib 資料夾。

### a 建立 lib 資料夾。

```
mkdir lib
```

### b 安裝要求模組。

```
pip3 install requests -t lib/
```

## 6 將要求模組新增至 handler.py 指令碼。

```
import requests

def handler(context, inputs):
    print('Hello, your inputs were ' + inputs)
    return None
```

## 7 建立對 vRealize Automation 專案 API 的 GET 要求。

```
token = ''
vRAUrl = ''
r = requests.get(vRAUrl + '/iaas/api/projects', headers={'Authorization': 'Bearer ' + token})

print('Got response ' + r.text)
```

**8** 定義 token 和 vRAUrl 值。

- a 使用 vRealize Automation 身分識別服務 API 擷取存取 Token。請參閱〈[取得 vRealize Automation API 的存取 Token](#)〉
- b 對於 vRAUrl 值，請定義指令碼，讓其使用具有相同名稱的 vRealize Orchestrator 輸入參數。

```
vRAUrl = inputs["vRAUrl"]
```

- c 將新值新增至 handler.py 檔案。

```
import requests

def handler(context, inputs):
    token = 'ACCESS_TOKEN'
    vRAUrl = inputs["vRAUrl"]

    r = requests.get(vRAUrl + '/iaas/api/projects', headers={'Authorization': 'Bearer ' + token})

    print('Got response ' + r.text)

    return r.json()
```

**備註** 由於 vRealize Automation 專案 API 的回應以 JSON 格式傳回，因此請針對您的 vRealize Orchestrator 動作使用 Properties 或 Composite Type 傳回類型。

**9** 建立包含要求模組之 handler.py 檔案和 lib 資料夾的 ZIP 套件。

```
zip -r --exclude=*.zip -X vro-python-vra.zip .
```

## 後續步驟

將 PowerShell 指令碼匯入 vRealize Orchestrator 動作。請參閱在 [vRealize Orchestrator 用戶端中建立動作](#)。

**建立呼叫 vRealize Automation 專案 API 的 Node.js 指令碼**

建立使用 Node.js 呼叫 vRealize Automation 專案 API 的範例指令碼。

## 必要條件

下載並安裝 Node.js 14。請參閱〈[Node.js 下載](#)〉。

## 程序

- 1 在本機電腦上，開啟命令列 Shell。
- 2 建立 vro-node-vra 資料夾。

```
mkdir vro-node-vra
```



### 3 導覽至 vro-node-vra 資料夾。

```
cd vro-node-vra
```

### 4 建立名為 handler.js 的 Node.js 指令碼。

```
touch handler.js
```

handler.js 指令碼必須定義一個接受兩個引數、vRealize Orchestrator 工作流程執行的內容和繫結的 vRealize Orchestrator 輸入的函數。

```
exports.handler = (context, inputs) => {
  console.log('Hello, your inputs were ' + inputs);
  return null;
}
```

**備註** 透過使用標準記錄程式庫，您在使用指令碼的動作中記錄的所有內容也會顯示在工作流程記錄中。必須在 vRealize Orchestrator 用戶端中為指令碼的輸入和傳回設定對應的輸入參數和傳回類型。例如，指令碼中的 vRAUrl 輸入必須在 vRealize Orchestrator 用戶端中具有名為 vRAUrl 的對應輸入參數。同樣地，如果您的指令碼傳回字串值，則 vRealize Orchestrator 用戶端中設定的傳回類型也必須是字串類型。如果動作傳回複雜物件，您可以使用 Properties 或 Composite Type 傳回類型。

### 5 安裝 Node.js 要求模組。

```
npm install request
```

**重要** 第三方相依性模組必須安裝在主要 vro-node-vra 指令碼資料夾的根層級 node\_modules 資料夾中。請勿移動或重新命名此資料夾。

### 6 將要求模組新增至 handler.js 指令碼。

```
const request = require('request');

exports.handler = (context, inputs) => {
  console.log('Hello, your inputs were ' + inputs);
  return null;
}
```

### 7 建立對 vRealize Automation 專案 API 的 GET 要求。

```
const token = '';
const vRAUrl = '';
request.get(vRAUrl + '/iaas/api/projects', { 'auth': { 'bearer': token } }, function
(error, response, body) {
  console.log('Got response ' + body);
});
```

**8** 定義 token 和 vRAUrl 值。

- a 使用 vRealize Automation 身分識別服務 API 擷取存取 Token。請參閱〈[取得 vRealize Automation API 的存取 Token](#)〉。
- b 對於 vRAUrl 值，請定義指令碼，讓其使用具有相同名稱的 vRealize Orchestrator 輸入參數。

```
const vRAUrl = inputs.vRAUrl;
```

- c 新增值至 handler.js 檔案。

```
const request = require('request');
exports.handler = (context, inputs, callback) => {
  const vRAUrl = inputs.vRAUrl;
  const token = 'ACCESS_TOKEN';
  request.get(vRAUrl + '/iaas/api/projects', { 'auth': { 'bearer': token } },
    function (error, response, body) {
      console.log('Got response ' + body);
      callback(null, JSON.parse(body));
    });
}
```

**備註** 由於 vRealize Automation 專案 API 的回應以 JSON 格式傳回，因此請針對您的 vRealize Orchestrator 動作使用 Properties 或 Composite Type 傳回類型。

- 9** 建立包含要求模組之 handler.js 檔案和 node\_modules 資料夾的 ZIP 套件。

```
zip -r --exclude=*.zip -X vro-node-vra.zip .
```

## 後續步驟

將 Node.js 指令碼匯入 vRealize Orchestrator 動作。請參閱在 [vRealize Orchestrator 用戶端](#) 中建立動作。

**建立呼叫 vRealize Automation 專案 API 的 PowerShell 指令碼**

建立使用 PowerShell 呼叫 vRealize Automation 專案 API 的範例指令碼。

## 程序

- 1** 在本機電腦上，開啟命令列 Shell。
- 2** 建立 vro-powershell-vra 資料夾。

```
mkdir vro-powershell-vra
```

- 3** 導覽至 vro-powershell-vra 資料夾。

```
cd vro-powershell-vra
```

#### 4 建立名為 handler.ps1 的 PowerShell 指令碼。

```
touch handler.ps1
```

handler.ps1 指令碼必須定義一個接受兩個引數、vRealize Orchestrator 工作流程執行的內容和繫結的 vRealize Orchestrator 輸入的函數。

```
function Handler {
    Param($context, $inputs)

    $inputsString = $inputs | ConvertTo-Json -Compress
    Write-Host "Inputs were $inputsString"
}
```

**備註** 透過使用標準記錄程式庫，您在使用指令碼的動作中記錄的所有內容也會顯示在工作流程記錄中。必須在 vRealize Orchestrator 用戶端中為指令碼的輸入和傳回設定對應的輸入參數和傳回類型。例如，指令碼中的 vRAUrl 輸入必須在 vRealize Orchestrator 用戶端中具有名為 vRAUrl 的對應輸入參數。同樣地，如果您的指令碼傳回字串值，則 vRealize Orchestrator 用戶端中設定的傳回類型也必須是字串類型。如果動作傳回複雜物件，您可以使用 Properties 或 Composite Type 傳回類型。

#### 5 安裝 PowerShell 宣告模組。

**重要** 第三方相依性模組必須安裝在主要 vro-powershell-vra 指令碼資料夾的根層級資料夾中。在此使用案例中，您可以為宣告模組建立 Modules 資料夾。

##### a 建立 Modules 資料夾。

```
mkdir Modules
```

##### b 安裝宣告模組。

```
pwsh -c "Save-Module -Name Assert -Path ./Modules/ -Repository PSGallery"
```

#### 6 將宣告模組新增至 handler.ps1 指令碼。

```
Import-Module Assert

function Handler {
    Param($context, $inputs)

    $inputsString = $inputs | ConvertTo-Json -Compress
    Write-Host "Inputs were $inputsString"
}
```

#### 7 建立對使用 Invoke-RestMethod cmdlet 的 vRealize Automation 專案 API 的 GET 要求。

```
$token = ''
$vRAUrl = ''
```

```
$projectsUrl = $vRAUrl + "/project-service/api/projects"
$response = Invoke-RestMethod $projectsUrl + '/iaas/api/projects' -Headers
@{'Authorization' = "Bearer $token"} -Method 'GET'

Write-Host "Got response: $response"
```

## 8 定義 token 和 vRAUrl 值。

- a 使用 vRealize Automation 身分識別服務 API 擷取存取 Token。請參閱〈[取得 vRealize Automation API 的存取 Token](#)〉。
- b 新增 Assert-NotNull 和 Assert-Type 宣告模組屬性。

```
$token | Assert-NotNull
$token | Assert-Type String
```

- c 對於 vRAUrl 值，請定義指令碼，讓其使用具有相同名稱的 vRealize Orchestrator 輸入參數。

```
$vRAUrl = $inputs.vRAUrl
```

- d 新增值至 handler.ps1 檔案。

```
Import-Module Assert
$ErrorActionPreference = "Stop"
function Handler {
    Param($context, $inputs)
    $token = "ACCESS_TOKEN"
    $token | Assert-NotNull
    $token | Assert-Type String
    $vRAUrl = $inputs.vRAUrl
    $projectsUrl = $vRAUrl + "/project-service/api/projects"
    $response = Invoke-RestMethod $projectsUrl -Headers @{'Authorization' = "Bearer
$token"} -Method 'GET'

    Write-Host "Got response: $response"

    return $response
}
```

**備註** 由於 vRealize Automation 專案 API 的回應以 JSON 格式傳回，因此請針對您的 vRealize Orchestrator 動作使用 Properties 或 Composite Type 傳回類型。

## 9 建立包含宣告模組之 handler.ps1 檔案和 Modules 資料夾的 ZIP 套件。

```
zip -r --exclude=*.zip -X vro-powershell-vra.zip .
```

### 後續步驟

將 PowerShell 指令碼匯入 vRealize Orchestrator 動作。請參閱在 [vRealize Orchestrator 用戶端中建立動作](#)。

工作流程是一系列依序執行的動作和決定。**vRealize Orchestrator** 提供可執行一般管理工作的工作流程程式庫。**vRealize Orchestrator** 也提供工作流程執行的個別動作程式庫。

工作流程結合了動作、決策及結果，當以特定的順序執行時，即可在虛擬環境中完成特定工作或特定程序。工作流程會執行工作，例如佈建虛擬機器、備份、執行定期維護、傳送電子郵件、執行 SSH 作業、維護實體基礎結構，以及其他一般公用程式作業。工作流程可按照功能接受輸入。您能夠建立按照定義的排程執行的工作流程，或在預期的事件發生時執行的工作流程。您、其他使用者、其他工作流程或動作，或外部程序 (例如應用程式的 Web 服務呼叫)，均可提供資訊。工作流程會在執行前進行資訊的一些驗證和篩選。

工作流程能夠呼叫其他工作流程。例如，您可以讓工作流程呼叫其他工作流程，以建立新的虛擬機器。

您可使用 **vRealize Orchestrator Client** 介面的整合式開發環境 (IDE) 建立工作流程，藉以提供工作流程程式庫的存取權，以及對工作流程引擎執行工作流程的能力。工作流程引擎也能夠從您外掛於 **vRealize Orchestrator** 的外部程式庫取得物件。這項功能可用於自訂程序，或實作第三方應用程式提供的功能。

本章節討論下列主題：

- **vRealize Orchestrator** 工作流程程式庫中的標準工作流程
- 在 **vRealize Orchestrator** 用戶端中建立工作流程
- 從父系工作流程編輯工作流程和動作
- **vRealize Orchestrator** 輸入表單設計工具
- **vRealize Orchestrator** 用戶端中的使用者互動要求
- 在 **vRealize Orchestrator** 用戶端中排程工作流程
- 在工作流程中尋找物件參考

## **vRealize Orchestrator** 工作流程程式庫中的標準工作流程

**vRealize Orchestrator** 提供標準工作流程程式庫，可用於自動化虛擬基礎結構中的作業。標準程式庫中的工作流程皆鎖定在唯讀狀態。若要自訂標準工作流程，您必須複製該工作流程。可完全編輯的重複工作流程或您建立的自訂工作流程。

可透過以 HTML5 為基礎的 vRealize Orchestrator Client 的**程式庫 > 工作流程**功能表存取工作流程程式庫的內容。用戶端中的標準和自訂工作流程均會使用標籤進行組織整理。例如，您可以透過在工作流程程式庫搜尋方塊中輸入 **SSH**，存取**產生金鑰配對**工作流程。

---

**備註** 除非您複製工作流程，否則無法將標籤新增到標準工作流程。

---

## 在 vRealize Orchestrator 用戶端中建立工作流程

您可以使用 vRealize Orchestrator Client 來建立和編輯工作流程。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 選取**程式庫 > 工作流程**。
- 3 按一下**新增工作流程**。
- 4 輸入新工作流程的名稱，然後按一下**建立**。
- 5 可以使用工作流程編輯器設定變數、工作流程輸入和輸出、架構結構以及工作流程的呈現。
- 6 若要完成工作流程編輯，請按一下**儲存**。

---

**備註** 您可以在**版本歷程記錄**索引標籤中追蹤工作流程變更。如需詳細資訊，請參閱 [vRealize Orchestrator 物件的版本歷程記錄](#)。

---

### 後續步驟

您可以使用 vRealize Orchestrator Token 重新執行功能，以最佳化工作流程效能。如需詳細資訊，請參閱 [在 vRealize Orchestrator 用戶端中使用工作流程 Token 重新執行](#)。

## 從父系工作流程編輯工作流程和動作

在 vRealize Orchestrator Client 中，直接從父系工作流程編輯工作流程和動作。

直接從父系工作流程編輯子系工作流程和動作，有助於簡化工作流程開發。

### 必要條件

建立呼叫另一個工作流程和/或動作的工作流程。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 工作流程**，然後選取您的工作流程。
- 3 選擇**配置**索引標籤。
- 4 根據物件類型，在工作流程畫布中，按兩下**工作流程元素**或**動作元素**。
- 5 編輯物件。

- 6 若要完成子系工作流程或動作的編輯，請按一下**儲存**。
- 7 若要返回父系工作流程，請關閉物件編輯器。

## vRealize Orchestrator 輸入表單設計工具

如果工作流程需要輸入參數，將會開啟一個對話方塊以供使用者輸入所需的值。您可以使用輸入表單設計工具來組織整理此對話方塊的內容、配置和呈現。

輸入表單設計工具位於工作流程編輯器的**輸入表單**索引標籤中。此設計工具由導覽功能表、設計畫布和內容功能表組成。您可以將輸入和一般元素從左側功能表拖曳至設計畫布。在畫布中，您可以設定輸入參數的位置，將它們組織整理成不同的輸入索引標籤，並設定輸入參數內容。

---

**備註** 在輸入表單設計工具中，不可使用工作流程編輯器的**變數**索引標籤中的內容。僅可以使用**輸入/輸出**索引標籤中的參數。

---

### 一般元素

您可以將下拉式功能表和密碼文字方塊等一般元素新增至輸入表單設計工具。一般元素不會對應到實際輸入參數，但可繫結至輸入參數。

## 在 vRealize Orchestrator 用戶端中建立工作流程輸入參數對話方塊

您可以使用輸入表單設計工具來建立和自訂工作流程輸入參數對話方塊。

### 必要條件

確認工作流程含有輸入參數的定義清單。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 工作流程**。
- 3 選取自訂工作流程。
- 4 按一下**輸入表單**索引標籤。
- 5 (選擇性) 建立索引標籤，以便在輸入對話方塊中使用。  
您可以使用索引標籤來組織整理對話方塊的結構。
- 6 選取您的輸入參數。
- 7 編輯輸入參數的內容。  
如需有關輸入參數內容的詳細資訊，請參閱 [vRealize Orchestrator 用戶端中的輸入參數內容](#)。
- 8 (選擇性) 將一般元素新增至畫布，並將其繫結到輸入參數。
- 9 (選擇性) 為輸入參數新增外部驗證。如需詳細資訊，請參閱 [使用動作驗證 vRealize Orchestrator 工作流程輸入](#)。
- 10 按一下**儲存**。

## 結果

您已建立工作流程對話方塊的配置並設定輸入參數的內容。

## vRealize Orchestrator 用戶端中的輸入參數內容

您可以設定參數內容，限制使用者執行 vRealize Orchestrator 工作流程時提供的輸入參數。

透過 vRealize Orchestrator，您可以定義用於量化工作流程中使用的輸入參數值的參數內容。您定義的參數內容會限制使用者可在 vRealize Orchestrator 工作流程中提供的輸入參數的類型和值。

參數內容會驗證輸入參數，並修改文字方塊在輸入參數對話方塊中的顯示方式。部分參數內容可建立參數之間的相依性。

參數內容	說明
標籤	設定輸入參數標籤。
顯示類型	設定輸入文字方塊顯示類型。
可見度	設定輸入參數的可見度。
唯讀	將輸入文字方塊設為唯讀。
自訂說明	設定輸入參數路標說明。
預設值	設定輸入參數的預設值。
步階	用於數字類型輸入。根據每按一下增加的輸入參數值進行設定。
必填	設定輸入參數值是否為必填。
規則運算式	使用規則運算式驗證輸入。
最小值	設定參數的最小值或長度。
最大值	設定參數的最大值或長度。
符合文字方塊	設定輸入參數值，以與其他輸入參數的值相符。
值來源	在 <b>外觀</b> 、 <b>值</b> 和 <b>限制</b> 索引標籤中設定參數內容的值來源。 <b>備註</b> 您可以使用 <b>外部來源</b> 以匯入外部動作的值。篩選可用動作是按參數類型完成的。

## 使用動作驗證 vRealize Orchestrator 工作流程輸入

使用外部動作驗證自訂工作流程的輸入。

### 必要條件

使用輸入參數建立自訂工作流程。如需詳細資訊，請參閱 [在 vRealize Orchestrator 用戶端中建立工作流程](#)。

您可以使用輸入表單設計工具，為工作流程輸入建立外部驗證。外部驗證使用的動作指令碼會在輸入參數值包含錯誤時傳回字串值。如果輸入參數值有效，外部驗證不會傳回任何內容。



## 程序

### 1 登入 vRealize Orchestrator 用戶端。

### 2 建立驗證動作。

- a 導覽至**程式庫 > 動作**。
- b 按一下**新增動作**。
- c 在**摘要**索引標籤上輸入所需的資訊。
- d 輸入驗證動作輸入參數。

---

**備註** 驗證動作輸入參數的名稱必須與正在驗證的工作流程輸入參數的名稱相同。

---

- e 在**指令碼**索引標籤上輸入驗證動作的指令碼。

```
if (in_1=="invalid") {
    return "in_1 can't be invalid!";
}

if (in_2=="invalid") {
    return "in_2 can't be invalid!";
}

//inputs are valid, return nothing
```

---

**備註** 先前的指令碼是簡單的範例，並不代表可以使用的驗證指令碼的完整範圍。

---

- f 按一下**儲存**。

### 3 套用外部驗證。

- a 導覽至**程式庫 > 工作流程**。
- b 選取自訂工作流程。
- c 選取**輸入表單**索引標籤。
- d 選取畫面左上方的剪貼簿圖示。
- e 將 vRealize Orchestrator 驗證元素拖曳至畫布。
- f 選取驗證元素，輸入驗證標籤，然後選取驗證動作。
- g (選擇性) 建立其他驗證元素。
- h 按一下**儲存**。

### 4 執行工作流程。

如果驗證發生錯誤，則會傳回字串。如果驗證成功，則驗證不會傳回任何內容，並繼續執行工作流程。

## 結果

您已為自訂 vRealize Orchestrator 工作流程建立外部驗證。

## vRealize Orchestrator 用戶端中的使用者互動要求

在工作流程完成之前，可能會要求額外的使用者輸入。

需要進一步使用者互動的工作流程會暫停作業，直到使用者提供要求的輸入參數。工作流程會定義哪些使用者可提供要求的資訊，並據此傳送互動的要求。等待使用者輸入的工作流程會顯示在 vRealize Orchestrator Client 儀表板的**最近的工作流程執行**面板中和右上方的通知功能表中。

## 在 vRealize Orchestrator 用戶端中排程工作流程

您可以使用排程，將 vRealize Orchestrator 工作流程執行自動化。

排程工作流程執行時，可設定排定工作的執行日期、時間以及間隔。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 從**程式庫**功能表中選取您的工作流程，然後在 [工作流程] 面板中，按一下**排程**。
- 3 在**一般**、**排程**和**工作流程**類別中設定排定的工作參數。

**備註** 只有在需要輸入參數的工作流程中，才會看到**工作流程參數**類別。

參數	說明
名稱	排定工作的名稱。
說明	簡短的說明，詳述排定工作的目的。
開始時間	工作流程第一次排定執行的日期與時間。
如果為過去時間則開始	如果排定的時間在過去，選擇是否要開始工作流程。 <b>是</b> ，會立即開始排定的工作流程。 <b>否</b> ，會在下一個排定週期開始工作流程。
排程	設定排定工作的週期模式和事件觸發項目。
結束日期	僅在選取 <b>無週期</b> 時才可見。設定排定工作結束的日期和時間。
工作流程	輸入工作流程的輸入參數。

- 4 按一下**建立**。

結果

您已為工作流程建立排定的工作。排定的工作流程會出現在**活動 > 已排定**中。您可以按一下排程面板中的**刪除**，以刪除排定的工作。

## 在 vRealize Orchestrator 用戶端中編輯排定的工作

您可以編輯排定的工作以變更參數，例如排定工作流程的日期、時間與週期。

#### 必要條件

建立排定工作流程的工作。

#### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 從**活動 > 已排定**中，選取您排定的工作。
- 3 在工作流程面板上，按一下**編輯**。
- 4 編輯排程，然後按一下**儲存**。

---

**備註** 建立排定的工作時，設定的輸入參數為唯讀狀態，且無法進行編輯。若要變更這些參數，請為此工作流程建立新的排定工作。

---

## 在工作流程中尋找物件參考

身為工作流程開發人員，您可以使用物件參考資訊將開發生命週期最佳化。

透過 vRealize Orchestrator Client，您可以尋找物件參考資訊。此功能有兩種作用：

- **尋找相依性**：在工作流程中尋找物件相依性的相關資訊。相依性可包括其他工作流程、動作、資源元素和組態元素。
- **尋找使用量**：瞭解選取的工作流程是否用於 vRealize Orchestrator Client 程式庫中的其他工作流程。

您可以從工作流程編輯器，或在卡視圖、清單視圖或樹狀結構視圖中，從 vRealize Orchestrator Client 程式庫存取物件參考的相關資訊。若要進一步瞭解 vRealize Orchestrator Client 程式庫不同類型的內容組織，請參閱 [vRealize Orchestrator Client 中的內容組織整理](#)。

下列程序將說明如何從工作流程編輯器存取物件參考。

#### 必要條件

開發至少包含一個物件參考的工作流程。

#### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 工作流程**，然後選取您的工作流程。
- 3 若要尋找物件相依性的相關資訊，請按一下**尋找相依性**。

---

**備註** 在相依性快顯視窗中，您可以從清單中選取參考的物件。選取某個物件後會開啟個別的 vRealize Orchestrator Client 索引標籤，您可以在其中檢視所選物件的詳細資料，或加以編輯。

---

- 4 若要尋找使用所選工作流程位置的相關資訊，請按一下**尋找使用量**。

您可以透過新增動作指令碼來修改 vRealize Orchestrator 工作流程。

vRealize Orchestrator Client 提供預先定義動作的程式庫以及自訂動作指令碼的動作編輯器。動作代表您在工作流程中用作建置區塊的個別函數。

動作是 JavaScript 函數。動作可取得多個輸入參數並擁有單一傳回數值。動作可呼叫 vRealize Orchestrator API 中的任何物件，或您使用外掛程式匯入到 vRealize Orchestrator 之任何 API 中的物件。

執行工作流程時，動作會從工作流程的變數取得輸入參數。這些變數可以是工作流程的初始輸入參數，或是工作流程中的其他元素在執行時設定的變數。

動作編輯器包含指令碼自動完成功能，以及具有可用指令碼類型及其說明文件的 API Explorer。

本章節討論下列主題：

- 在 vRealize Orchestrator 用戶端中建立動作
- 執行和偵錯動作
- Python、Node.js 和 PowerShell 指令碼的核心概念
- Python、Node.js 和 PowerShell 指令碼的執行階段限制

## 在 vRealize Orchestrator 用戶端中建立動作

您可以使用 vRealize Orchestrator Client 建立、編輯和刪除動作指令碼。

建立動作時，您可以使用下列執行階段：

- Python 3.7
- Node.js 14
- PowerCLI 11/Powershell 6.2
- PowerCLI 12.3.0/Powershell 7.1

---

**備註** PowerCLI 執行階段包括 PowerShell 和下列模組：VMware.PowerCLI、PowerNSX、PowervRA。

---

## 必要條件

在建立 Python、Node.js 或 PowerShell 指令碼之前，請確認您熟悉開發使用這些執行階段之 vRealize Orchestrator 相容指令碼的核心概念。請參閱 [Python、Node.js 和 PowerShell 指令碼的核心概念](#)。

## 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至 **程式庫 > 動作**。
- 3 按一下 **新增動作**。
- 4 在 **一般** 索引標籤上，輸入動作的名稱和模組名稱。

---

**備註** 每個動作的名稱和模組名稱必須是唯一的。動作名稱必須是有效的 JavaScript 函數。動作名稱必須為單一字組，其中只能包含字母、數字、貨幣符號（「\$」）和底線（「\_」）符號。模組名稱必須由以點（「.」）字元分隔的字組組成。

---

- 5 （選擇性）建立動作的說明、版本號碼、標籤和群組權限。
- 6 在 **指令碼** 索引標籤上，新增動作輸入，選取輸出的傳回類型，並撰寫指令碼。

---

**備註** 透過從 **類型** 下拉式功能表中選取 **Zip**，您可以匯入外部指令碼來源及其相依性模組（如果適用）。

---

- 7 若要完成動作編輯，請按一下 **儲存**。

隨即出現指示動作已儲存的訊息。

## 後續步驟

若要檢視如何使用 vRealize Orchestrator 動作的使用案例範例，請參閱 [如何使用 Python 在 vRealize Orchestrator 中整合 Amazon Web Services](#)。

# 執行和偵錯動作

您可以透過從動作編輯器直接執行並偵錯來改善動作。

您可以從 vRealize Orchestrator 用戶端的動作編輯器直接執行和偵錯動作。透過此功能，您可以保證動作在整合到工作流程時按預期執行。

## 在 vRealize Orchestrator 用戶端中執行動作

做為工作流程設計人員，您需要先執行動作，然後再將其整合至工作流程。

## 必要條件

建立動作。請參閱 [在 vRealize Orchestrator 用戶端中建立動作](#)。

## 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至 **程式庫 > 動作**，然後選取要執行的動作。

3 按一下**執行**。

4 輸入所需的輸入參數，然後按一下**執行**。

動作執行完成後，按一下**結果/輸入**索引標籤。如果動作執行發生錯誤，將以紅色顯示在此索引標籤上。您可以從**動作結果**元素中檢視動作執行的詳細資料。

**備註** 將不會儲存動作執行的結果。

## 在 vRealize Orchestrator 用戶端中偵錯動作

做為工作流程設計人員，您可以透過在指令碼中插入中斷點來偵錯動作。

vRealize Orchestrator 包含內建的偵錯工具，可用來偵錯動作的指令碼和輸入內容。在動作編輯器中，可以透過在動作的指令碼行中插入中斷點來起始偵錯程序。

**備註** 內建偵錯工具僅適用於使用預設 JavaScript 執行階段的動作。如需如何使用不同執行階段來偵錯動作指令碼的範例，請參閱**偵錯 Amazon Web Services 動作**。

必要條件

建立動作。請參閱在 [vRealize Orchestrator 用戶端中建立動作](#)。

程序

1 登入 vRealize Orchestrator 用戶端。

2 導覽至**程式庫 > 動作**，然後選取要偵錯的動作。

3 在動作編輯器中，將中斷點新增至您要偵錯的動作指令碼的行。

4 按一下**偵錯**。

5 輸入動作的輸入參數，然後按一下**執行**。

在偵錯模式下執行的動作隨即開始。

6 到達中斷點後暫停動作執行時，請選取下列其中一個選項：

選項	說明
<b>繼續</b>	繼續執行動作，直到到達其他中斷點或動作執行完成為止。
<b>跳轉到</b>	跳轉到目前動作函數。如果偵錯工具無法深入到函數的目前行，它會執行 <b>跳過</b> 作業。
<b>跳過</b>	偵錯工具會繼續進入目前函數的下一行。
<b>返回執行</b>	偵錯工具會進入將在目前函數傳回時執行的行。

7 (選擇性) 在**偵錯工具**索引標籤上，新增運算式。

8 (選擇性) 在**偵錯工具**索引標籤上，編輯變數的值。

## Python、Node.js 和 PowerShell 指令碼的核心概念

建立用於 vRealize Orchestrator 的指令碼時，您必須確認指令碼具有正確的結構和格式。

### 支援的執行階段

對於開發 vRealize Orchestrator 動作和工作流程，您可以使用下列執行階段：

- Python 3.7
- Node.js 14
- PowerCLI 11/Powershell 6.2
- PowerCLI 12.3.0/Powershell 7.1

---

**備註** PowerCLI 執行階段包括 PowerShell 和下列模組：VMware.PowerCLI、PowerNSX、PowervRA。

---

您可以將任何自訂來源代碼新增至新的執行階段，但若要接受內容和輸入，以及傳回來自 vRealize Orchestrator 引擎的結果及將結果傳回此引擎，您必須遵循正確的函數格式。

### 指令碼處理建議

對於較簡單的指令碼工作，您可以將**可編寫指令碼的工作**元素新增至工作流程架構。您可以將 vRealize Orchestrator 動作用於較複雜的指令碼工作。

使用動作提供兩個具體的優點：

- 可以在工作流程以外單獨建立、更新、匯入和匯出動作。
- 動作是獨立物件，可在其本身的環境中執行和接受偵錯，進而產生更順暢的開發程序。請參閱[執行和偵錯動作](#)。

### 指令碼函數需求

指令碼函數的預設名稱為 **handler**。此函數接受兩個引數、內容和輸入。內容是包含系統資訊的對應物件。例如，vroURL 可以包含您想要呼叫的 vRealize Orchestrator 執行個體的 URL，而 executionId 包含工作流程執行的 Token 識別碼。

輸入是包含提供給動作的所有輸入的對應物件。例如，如果您在動作中定義了名為 myInput 的輸入，則可以從輸入引數 (例如 inputs.myInput 或 inputs["myInput"]) 存取此輸入，具體取決於您的執行階段。從函數傳回的任何內容都是動作的結果。因此，動作的傳回類型必須對應指令碼在 vRealize Orchestrator 中傳回的內容類型。如果傳回基本數字，則動作傳回類型必須為數字類型。如果傳回字串，則動作傳回類型必須為字串類型。如果傳回複雜物件，則傳回類型必須對應 Properties 或 Composite Type。這些相同的原則也適用於陣列。

Python、Node.js 和 PowerShell 執行階段支援的輸入和輸出參數類型：

- String
- Number

- Boolean
- Date
- Properties
- Composite Type

## 定義項目處理常式

依預設，項目處理常式的值為 `handler.handler`。該值表示 vRealize Orchestrator 引擎會在 ZIP 套件中尋找名為 `handler.py`、`handler.js` 或 `handler.ps1` 的頂層檔案，其中包含名為 `handler` 的函數。與函數和處理常式檔案名稱的任何差異都必須反映在項目處理常式的值中。例如，如果主要處理常式名為 `index.js`，而函數名為 `callMe`，則必須將項目處理常式的值設定為 **`index.callMe`**。

## 在外部 IDE 中偵錯執行階段指令碼

vRealize Orchestrator 支援在外部 IDE 中偵錯 Python 和 Node.js 指令碼。您無法在外部 IDE 中偵錯 PowerShell 指令碼。

## Python、Node.js 和 PowerShell 指令碼的執行階段限制

某些 Python、Node.js 或 PowerShell 指令碼會要求您變更 vRealize Orchestrator Client 中的記憶體和逾時值。

vRealize Orchestrator Client 將針對 Python、Node.js 和 PowerShell 動作指令碼使用一組預設記憶體和逾時值：

- 記憶體：64 MB
- 逾時：180 秒

如果您的動作指令碼超過其中一或兩個預設值，則動作執行會失敗。例如，動作指令碼可能會使用多個第三方相依性模組。在此情況下，**64 MB** 預設記憶體限制可能不足。

為避免因資源不足導致動作執行失敗，請從動作編輯器變更記憶體和逾時值。

---

**備註** 您也可以考慮將指令碼分為多個可新增至工作流程的可編寫指令碼工作元素。

---

### 程序

- 1 登入 vRealize Orchestrator Client。
- 2 導覽至**程式庫 > 動作**，然後選取動作。
- 3 選取**指令碼索引**標籤。
- 4 在**執行階段限制**下，變更記憶體和逾時值。
- 5 按一下**儲存**。
- 6 若要測試新的執行階段限制，請按一下**偵錯**。



# 管理組態元素

# 7

組態元素是一系列變數，您可以使用這些變數在整個 vRealize Orchestrator 伺服器部署中設定常數。

您可以使用組態元素，使變數可供 vRealize Orchestrator 伺服器上執行的所有工作流程、動作及原則使用。

如果您建立一個套件，且其中包含的工作流程、動作或原則使用某個組態元素中的變數，則 vRealize Orchestrator 會自動在套件中包含該組態元素。如果您將包含組態元素的套件匯入到另一個 vRealize Orchestrator 伺服器，您也可以匯入組態元素變數值。例如，如果您建立一個需要變數值的工作流程，且該值取決於執行工作流程所在的 vRealize Orchestrator 伺服器，則在組態元素中設定這些變數可讓您匯出該工作流程讓另一個 vRealize Orchestrator 伺服器使用。因此，組態元素可讓您更輕鬆地在伺服器之間交換工作流程、動作與原則。

---

**備註** 您無法從自 vRealize Orchestrator 5.1 或更早版本匯出的組態元素，匯入組態元素變數的值。

---

本章節討論下列主題：

- [在 vRealize Orchestrator 用戶端中建立組態元素](#)

## 在 vRealize Orchestrator 用戶端中建立組態元素

您可以使用組態元素在整個 vRealize Orchestrator 伺服器中設定通用變數。在伺服器中執行的所有元素會使用您在組態元素中設定的變數。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**資產 > 組態**。
- 3 選取**新增組態**。
- 4 輸入組態元素名稱。
- 5 選取**變數索引**標籤。

**6** 若要建立本機變數，請按一下**新增**。

- a 輸入變數名稱。
- b 選取變數類型。

---

**備註** 若要建立組態變數的陣列，請選取**陣列**核取方塊。

---

- c (選擇性) 輸入組態變數的值。
- d 按一下**儲存**。

**7** 若要完成建立組態元素，請按一下**儲存**。

後續步驟

您可以使用組態元素為工作流程、動作或原則提供變數。

原則是監視系統活動的事件觸發器。原則會回應由於特定 vRealize Orchestrator 物件的狀態或效能變更所發出的預先定義事件。

原則是在 vRealize Orchestrator 中或在 vRealize Orchestrator 透過外掛程式存取的技術中發生特定的預先定義事件時，執行特定工作流程或指令碼的一系列規則、量表、臨界值和事件篩選器。當原則正在執行時，vRealize Orchestrator 可以持續評估原則規則。對於執行個體，您能夠實作原則量表和臨界值來監視 VC:HostSystem 和 VC:VirtualMachine 類型的 vCenter Server 物件行為。

本章節討論下列主題：

- 在 vRealize Orchestrator 用戶端中建立和套用原則
- vRealize Orchestrator 用戶端中的原則元素
- 在 vRealize Orchestrator 用戶端中管理原則執行

## 在 vRealize Orchestrator 用戶端中建立和套用原則

您可以使用原則針對特定事件監控 vRealize Orchestrator 系統的活動。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 原則**。
- 3 選取**新增原則**。  
您已建立一個空白原則。
- 4 輸入原則名稱和版本號碼。
- 5 選取**變數**索引標籤。
- 6 若要建立本機變數，請按一下**新增**。
  - a 輸入變數名稱。
  - b 選取變數類型。

---

**備註** 若要建立原則變數的陣列，請選取**陣列**核取方塊。

---

- c 輸入變數值。

**備註** 若要匯入組態元素變數的值，您可以使用**繫結至組態**。

- d 按一下**儲存**。

- 7 在**定義**索引標籤上，新增原則元素並設定事件處理常式。

如需有關原則元素的詳細資訊，請參閱 [vRealize Orchestrator 用戶端中的原則元素](#)。

- 8 按一下**儲存**。

您已設定原則。

#### 後續步驟

若要啟動原則，請選取該原則並按一下**執行**。輸入原則執行名稱，並在出現提示時輸入所需的輸入參數。

若要檢視原則狀態，請導覽至**活動 > 原則執行**。

## vRealize Orchestrator 用戶端中的原則元素

您可以使用原則元素在事件發生時執行預先定義的 vRealize Orchestrator 工作流程或指令碼。

您可以新增原則元素，以便觸發工作流程或指令碼執行來回應由物件觸發的事件。透過定期事件元素，您可以排程工作流程或指令碼執行。透過根元素，您可以設定原則的啟動或停止行為。原則元素可以具有定義原則元素必須何時執行的事件處理常式。

**備註** 啟動原則元素的事件處理常式可以是工作流程或動作指令碼。如果您同時將工作流程與指令碼新增至事件處理常式，原則會忽略指令碼觸發器並僅使用工作流程觸發器。

事件處理常式	說明
<b>OnInit</b>	每次啟動原則時都會觸發原則元素。
<b>OnExit</b>	每次停止原則時都會觸發原則元素。
<b>OnExecute</b>	由定期事件元素使用。在定期事件元素中指定的期間內觸發原則元素。

**備註** 插入 vRealize Orchestrator 資料庫的技術可擁有唯一事件處理常式。例如，透過 **SNMP** 外掛程式，您可以在建立以 **SNMP** 為基礎的原則元素時，使用 **OnTrap** 事件處理常式。

原則元素在原則編輯視窗的**定義**索引標籤中設定。

## 在 vRealize Orchestrator 用戶端中管理原則執行

您可以使用 vRealize Orchestrator Client 管理有關何時重新啟動 vRealize Orchestrator 伺服器之原則的原則優先順序和伺服器啟動行為。

## 必要條件

建立和執行原則。如需詳細資訊，請參閱 [在 vRealize Orchestrator 用戶端中建立和套用原則](#)。

## 程序

**1** 以管理員身分登入 vRealize Orchestrator 用戶端。

**2** 導覽至**活動 > 原則執行**。

**3** 按一下您想要管理的原則執行。

**4** 按一下**停止**。

原則狀態隨即變更為**已停止**。

**5** 在**一般**索引標籤上，設定原則優先順序和伺服器啟動行為。

**6** 若要重新啟動原則，請按一下**執行**。

原則狀態隨即變更為**執行中**。

# 管理資源元素

# 9

工作流程可以使用您獨立於 vRealize Orchestrator 建立的物件做為屬性。若要將外部物件用作工作流程中的屬性，請將其做為資源元素匯入至伺服器。

vRealize Orchestrator 工作流程可做為資源元素使用的物件包括映像檔、指令碼、XML 範本、HTML 檔案等。在 vRealize Orchestrator 伺服器中執行的任何工作流程皆可使用您匯入至 vRealize Orchestrator 的所有資源元素。

將物件做為資源元素匯入至 vRealize Orchestrator 後，可以在單一位置對物件進行變更，並將這些變更自動傳播至所有使用此資源元素的工作流程。

資源元素的最大大小為 16 MB。

您可以匯入、匯出、還原、更新和刪除資源元素。

使用 vRealize Orchestrator Client 建立、匯出和匯入套件。套件可用來匯出工作流程物件，以在其他 vRealize Orchestrator 執行個體上使用。

套件可包含工作流程、動作、原則、組態元素或資源元素。

當您將元素新增至套件時，vRealize Orchestrator 會檢查相依性，並將任何相依的元素新增至套件。例如，如果您新增的工作流程使用動作或其他工作流程，vRealize Orchestrator 會將這些動作與工作流程新增至套件。

當您匯入套件時，伺服器會比較其內容之不同元素的版本，以比對本機元素。這樣的比較會顯示本機元素與所匯入元素之間版本的差異。使用者可決定是否匯入套件，也可以選取要匯入的特定元素。

對於在 vRealize Orchestrator Client 中建立的大多數物件 (除了資源元素之外)，套件是匯出和匯入這些物件的唯一方式。

套件會使用數位權利管理來控制接收伺服器如何使用套件的內容。vRealize Orchestrator 會簽署與加密套件，以保護資料。套件使用 X509 憑證來追蹤哪些使用者匯出與重新散佈憑證。

## 在 vRealize Orchestrator 用戶端中建立套件

您可以使用套件匯出和匯入工作流程、原則、動作、外掛程式參考、資源元素及組態元素。系統會自動將與套件物件相關的所有相依元素新增至套件，以確保版本之間的相容性。若要刪除相依元素，您必須先移除相關的套件物件。

對於在 vRealize Orchestrator Client 中建立的大多數物件 (除了資源元素之外)，套件是匯出和匯入這些物件的唯一方式。

### 必要條件

確認 vRealize Orchestrator 伺服器包含工作流程、動作及原則等可新增至套件的物件。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**資產 > 套件**。
- 3 按一下**新增套件**。

- 4 在一般索引標籤上，輸入套件的名稱與說明。

**備註** 命名 vRealize Orchestrator Client 中的套件時不能使用特殊字元。

- 5 在內容索引標籤上，按一下**新增**。
- 6 選取您想要新增至套件的物件，然後按一下**新增**。

**備註** 在套件建立期間，相依元素會自動新增至套件但不會顯示在內容索引標籤中。若要檢視相依元素，請在套件建立之後選取內容索引標籤。

- 7 若要完成套件建立，請按一下**建立**。

## 在 vRealize Orchestrator 用戶端中匯出套件

您可以使用 vRealize Orchestrator Client 將套件匯出到另一個 vRealize Orchestrator 環境。

必要條件

建立包含您要匯出之 vRealize Orchestrator 物件的套件。如需詳細資訊，請參閱 [在 vRealize Orchestrator 用戶端中建立套件](#)。

程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**資產 > 套件**。
- 3 在套件上按一下**匯出**。
- 4 (選擇性) 選取其他匯出選項。

選項	說明
新增組態屬性值至套件	匯出組態元素的屬性值。
新增組態 <b>SecureString</b> 屬性值至套件	匯出 SecureString 組態屬性值。
新增全域標籤至套件	匯出全域標籤。

- 5 為匯入套件的使用者設定存取權限。

選項	說明
檢視內容	使用者可以檢視套件內容。
新增至套件	使用者可以將已匯入套件中的內容新增到其他套件。
編輯內容	使用者可以編輯套件內容。

- 6 按一下**確定**。

**備註** 副檔名為 .package 的檔案，會儲存到本機電腦的預設資料夾中。若要設定自訂資料夾，您可以變更瀏覽器中的儲存設定。



## 結果

您已匯出套件。您現在可以在另一個 vRealize Orchestrator 環境中使用匯出的物件。

## 在 vRealize Orchestrator 用戶端中匯入套件

使用 vRealize Orchestrator Client 匯入工作流程套件。透過匯入套件，您可以在另一個伺服器上重複使用來自某個 vRealize Orchestrator 伺服器的物件。

### 必要條件

- 備份您修改過的所有標準 vRealize Orchestrator 物件。
- 在遠端伺服器上，建立和匯出包含您要匯入之物件的套件。

### 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**資產 > 套件**。
- 3 按一下**匯入**，瀏覽至您要匯入的 .package 檔案，然後按一下**開啟**。
- 4 檢閱已匯入套件的資訊。
  - a **一般**索引標籤包含匯入套件的相關資訊，例如名稱、說明、包含的項目數量以及憑證資訊。  
系統可能會顯示提示，指出您需要信任來源 vRealize Orchestrator 執行個體的發行者憑證，然後才能匯入此檔案。
  - b **套件元素**索引標籤會列出匯入檔案中包含的物件。若套件中的物件版本比伺服器的版本新，系統會選取該物件版本來匯入。較舊版本的 vRealize Orchestrator 元素必須手動選取。
  - c 如果您不想從套件匯入組態元素的屬性值，請取消選取**匯入組態屬性值**。
  - d 從下拉式功能表中選取是否要匯入標籤。
- 5 按一下**匯入**。

# vRealize Orchestrator 用戶端中的 疑難排解

# 11

您可以使用度量、Token 重新執行、驗證和偵錯來疑難排解並監控 vRealize Orchestrator 執行個體。

本章節討論下列主題：

- vRealize Orchestrator 用戶端中的度量資料
- 在 vRealize Orchestrator 用戶端中使用工作流程 Token 重新執行
- 驗證 vRealize Orchestrator 工作流程
- 在 vRealize Orchestrator 用戶端中偵錯工作流程指令碼
- 依架構元素偵錯工作流程
- 為 Python 套件設定 Photon OS 容器

## vRealize Orchestrator 用戶端中的度量資料

vRealize Orchestrator 管理員可使用工作流程剖析和系統儀表板度量對 vRealize Orchestrator 系統和工作流程進行疑難排解。

剖析功能會收集工作流程執行的相關度量資料。工作流程剖析預設為啟用。您可以在**控制中心 > 延伸內容 > Profiler 8.6.0** 中停用自動剖析功能。

vRealize Orchestrator Client 中的度量資料的其他來源是系統儀表板，可提供系統層級度量。如需詳細資訊，請參閱 [使用 vRealize Orchestrator 系統儀表板](#)。

## 在 vRealize Orchestrator 用戶端中剖析工作流程

您可以剖析工作流程執行，以疑難排解及最佳化 vRealize Orchestrator 環境。

您可以使用 vRealize Orchestrator Client 的剖析功能，收集有關工作流程執行的實用度量資料。此資料可用於最佳化工作流程的效能。依預設，會自動剖析工作流程執行。您可以從 vRealize Orchestrator 控制中心的**延伸內容**頁面停用自動剖析，並手動執行剖析工具。若要手動執行剖析，請在程式庫中找到您的工作流程，並選取**動作 > 設定檔**。

必要條件

執行工作流程。

## 程序

- 1 登入 vRealize Orchestrator 用戶端。
- 2 導覽至**活動 > 工作流程執行**。
- 3 選取工作流程執行。

在工作流程執行架構上，您可以查看個別工作流程項目的相關資料。資料包括執行持續時間總計、最大持續時間和項目執行數目。您可以從頁面右上方的下拉式功能表中篩選此資訊。

- 4 選取**效能索引**標籤。

此索引標籤為您提供有關工作流程執行 CPU 時間、執行持續時間、Token 大小以及工作流程項目資料的度量資料。

---

**備註** 如果工作流程執行已暫停，例如工作流程正等待進一步輸入時，CPU 時間度量僅會擷取完成前發生的執行階段執行緒。

---

## 後續步驟

使用透過剖析收集的資料來最佳化您的工作流程。

## 使用 vRealize Orchestrator 系統儀表板

做為管理員，您可以使用 vRealize Orchestrator Client 系統儀表板，收集關於 vRealize Orchestrator 環境節點的有用度量資料。

您可以透過按一下 vRealize Orchestrator Client 儀表板頁面上方的**系統**索引標籤來存取系統儀表板。提供的資料包括：

- 節點狀態
- 節點內容
- 叢集設定。只能從系統儀表板檢視叢集設定。若要變更這些設定，請前往 vRealize Orchestrator 控制中心的 **Orchestrator 叢集管理** 頁面。
- 執行緒資訊
- 堆積記憶體
- 非堆積記憶體
- 檔案系統使用
- 驗證資料
- Orchestrator 資料庫連線集區
- 程序輸入引數

此資料可用於監控 vRealize Orchestrator 環境的個別節點的狀態以及疑難排解問題。若要在個別節點之間進行導覽，請按一下與系統儀表板上方的節點相關聯的索引標籤。

## 在 vRealize Orchestrator 用戶端中使用工作流程 Token 重新執行

您可以使用 Token 重新執行功能來檢視工作流程執行中的項目之間的轉換。

Token 重新執行功能記錄了工作流程項目之間每次轉換的內容資訊。對於每個工作流程項目，Token 重新執行功能會記錄工作流程執行的開始和結束時間，以及在工作流程項目執行結束後變更了哪些變數。

Token 重新執行功能也會參考針對每個工作流程項目產生的指令碼記錄訊息。

---

**備註** 工作流程項目轉換的相關資料會儲存在 vRealize Orchestrator PostgreSQL 資料庫中。刪除工作流程執行時，會從資料庫中移除此資料。

---

### 必要條件

- 從控制中心啟用 Token 重新執行功能。
  - a 以**根使用者**身分登入控制中心。
  - b 選取**延伸內容**。
  - c 按一下 **tokenreplay-8.6.0**。
  - d 若要啟用 Token 重新執行功能，請按一下**啟用**。
  - e 按一下**儲存**。

---

**備註** vRealize Orchestrator 伺服器可能需要最多 5 分鐘來重新整理延伸。

---

- 執行工作流程。

---

**備註** 依預設並不會對 vRealize Orchestrator 伺服器上的所有工作流程執行自動執行 Token 重新執行。您可以個別執行每個工作流程的 Token 重新執行，或從控制中心的**延伸內容**頁面，為所有工作流程啟用 Token 重新執行延伸。

---

### 程序

- 1 (選擇性) 為 vRealize Orchestrator 伺服器上的所有工作流程執行啟用 Token 重新執行。

---

**備註** 若要在未從控制中心啟用該功能的情況下執行個別的 Token 重新執行，請按一下工作流程編輯器頁面上的**使用重新執行來執行**。

---

- a 以**根使用者**身分登入控制中心。
- b 選取**延伸內容**。
- c 按一下 **tokenreplay-8.6.0**。
- d 若要為所有工作流程啟用 Token 重新執行功能，請確認已啟用**記錄所有工作流程執行的重新執行**。
- e 按一下**儲存**。

---

**備註** vRealize Orchestrator 伺服器可能需要最多 5 分鐘來重新整理延伸。

---

- 2 以管理員身分登入 vRealize Orchestrator 用戶端。
- 3 導覽至**活動 > 工作流程執行**。
- 4 選取工作流程執行。
- 5 從左側功能表選取工作流程執行項目。

**變數和記錄**索引標籤現在會顯示該工作流程項目的特定資訊。

## 驗證 vRealize Orchestrator 工作流程

vRealize Orchestrator 提供工作流程驗證工具。驗證工作流程有助於識別工作流程中的錯誤，並檢查資料是否從一個元素正確傳送到另一個元素。

依預設，vRealize Orchestrator 一律在您執行工作流程時執行工作流程驗證。

您驗證工作流程時，驗證工具會建立任何錯誤或警告的清單。檢查清單中的錯誤可找出包含錯誤的工作流程元素。

如果您在工作流程編輯器中執行驗證工具，工具將針對偵測到的錯誤提供建議的快速修正。部分快速修正需要額外資訊或輸入參數。其他快速修正則會為您解決錯誤。

工作流程驗證會檢查資料繫結和元素之間的連線。工作流程驗證不會檢查工作流程中的每個元素所執行的資料處理。因此，有效的工作流程可能會錯誤執行，而在架構元素中的函數不正確時產生錯誤的結果。

## 在 vRealize Orchestrator 用戶端中驗證工作流程並修正驗證錯誤

您必須先驗證工作流程，才能執行工作流程。只有在您開啟工作流程以進行編輯時，才能修正驗證錯誤。

### 必要條件

確定您有可驗證的完整工作流程，其中已連結架構元素，而且已定義繫結。

### 程序

- 1 以管理員身分登入 vRealize Orchestrator 用戶端。
- 2 導覽至**程式庫 > 工作流程**，然後選取您想要驗證的工作流程。
- 3 按一下**編輯**。
- 4 從頂端功能表按一下**驗證**。

如果工作流程有效，將出現確認訊息。如果工作流程無效，則會出現錯誤的清單。

- 5 對於無效的工作流程，請按一下錯誤訊息並採取適當步驟，以解決此問題。

驗證工具會為工作流程新增紅色圖示，指出發生錯誤的架構元素。驗證工具會盡可能顯示快速修正動作。

- 如果您同意建議的快速修正動作，請按一下工作流程來執行該動作。

- 如果您不同意建議快速修正動作，請關閉 [工作流程驗證] 對話方塊並手動修正架構元素。

---

**重要** 務必檢查 vRealize Orchestrator 建議的修正是否適當。

---

例如，建議的動作可能是刪除未使用的屬性，不過，實際上並未將屬性正確繫結。

- 6** 重複前述步驟，直到您消除所有的驗證錯誤為止。

結果

此時您即已驗證工作流程並修正驗證錯誤。

後續步驟

您可以執行工作流程。

## 在 vRealize Orchestrator 用戶端中偵錯工作流程指令碼

您可以透過在工作流程項目的指令碼中插入中斷點來偵錯工作流程執行。

到達中斷點後，您有多個選項可選擇繼續偵錯程序。在偵錯工作流程架構中的元素時，您可檢視有關工作流程執行的一般資訊、修改工作流程變數、新增要監視的運算式及檢視記錄訊息。

---

**備註** 在非生產環境中執行所有指令碼偵錯。

---

程序

- 1 以管理員身分登入 vRealize Orchestrator 用戶端。
- 2 從程式庫中選取工作流程。
- 3 開啟工作流程架構，選取工作流程元素，然後按一下**指令碼**索引標籤。
- 4 若要插入中斷點，請按一下行號左側的紅色圓圈。

---

**備註** 您只能使用指令碼在工作流程元素中插入中斷點。

---

- 5 若要在偵錯模式下執行工作流程，請按一下**偵錯**。

若工作流程需要輸入參數，您必須提供。

- 6 到達中斷點後暫停工作流程執行時，請選取其中一個可用選項。

選項	說明
繼續	繼續執行工作流程，直到到達其他中斷點或工作流程執行完成為止。
跳轉到	您可以使用此選項跳轉到某個工作流程元素。您無法在偵錯工作流程編輯器中的工作流程時，跳轉到巢狀工作流程元素。
跳過	跳過架構中的目前元素，並暫停下一個元素的工作流程執行。

---

**備註** 您可以透過按一下目前中斷點，指示偵錯工具忽略此中斷點。這會將中斷點符號變更為綠色三角形。

---

- 7 (選擇性) 在**偵錯工具**索引標籤上，插入要監視的運算式。

您可以使用運算式追蹤特定變數的完成情況。

- 8 (選擇性) 在**偵錯工具**索引標籤上，修改變數的值。

## 依架構元素偵錯工作流程

做為工作流程設計人員，您可以偵錯個別架構元素。

程序

- 1 登入 vRealize Orchestrator Client。
- 2 導覽至**程式庫 > 工作流程**，然後選取您的工作流程。
- 3 選擇**配置**索引標籤。
- 4 選取您要偵錯的工作流程元素，然後按一下元素左上方的偵錯按鈕。

**備註** 透過將中斷點新增至**工作流程元素**架構元素，您可以直接從父系工作流程偵錯子系工作流程。當偵錯工具到達**工作流程元素**架構元素時，它會開啟子系工作流程的架構視圖。

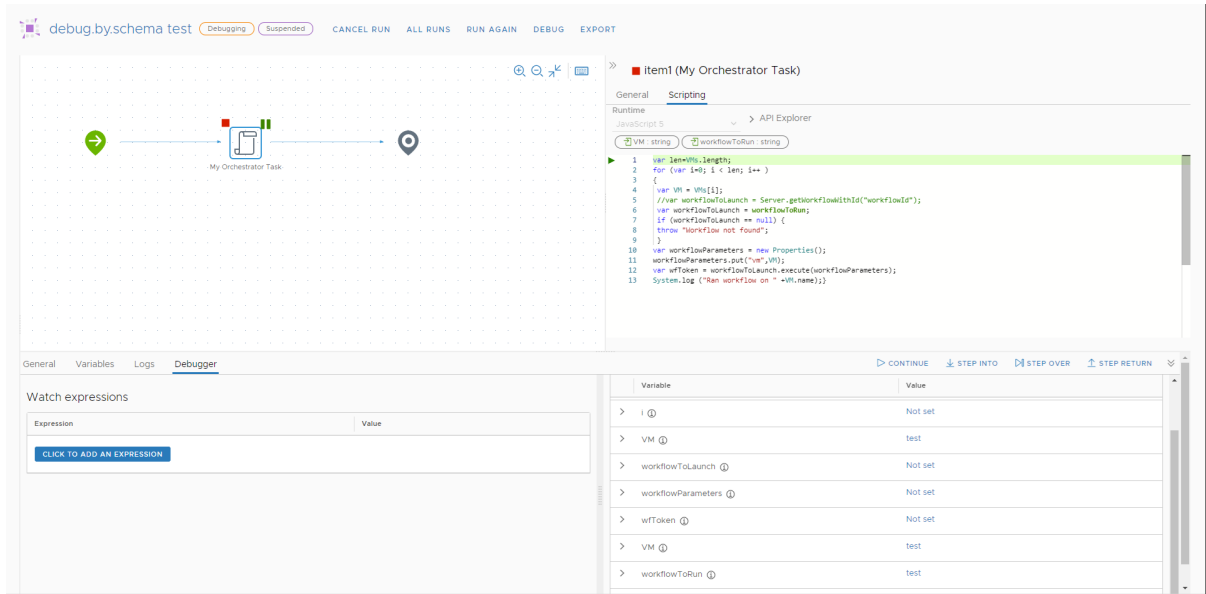
- 5 針對您要偵錯的任何其他架構元素重複以上步驟。
- 6 按一下**偵錯**。
- 7 輸入要求的輸入參數值，然後按一下**執行**。

工作流程執行隨即開始，並且在偵錯工具到達具有中斷點的架構元素時暫停。

- 8 在中斷點處，選取下列其中一個選項：

選項	說明
<b>繼續</b>	繼續執行工作流程，直到到達其他中斷點或工作流程執行完成為止。
<b>跳轉到</b>	跳轉到目前工作流程函數。如果偵錯工具無法深入到函數的目前行，它會執行 <b>跳過</b> 作業。
<b>跳過</b>	偵錯工具會繼續進入目前函數的下一行。
<b>返回執行</b>	偵錯工具會進入將在目前函數傳回時執行的行。

## 9 (選擇性) 在變數索引標籤上，編輯工作流程變數的值。



## 為 Python 套件設定 Photon OS 容器

視用於編譯 Python 指令碼的作業系統 (OS) 而定，在將相關的 ZIP 封存檔匯入至 vRealize Orchestrator 用戶端後，您的工作流程或動作可能會失敗。

在 vRealize Orchestrator 中用於 Python 的執行階段容器作業系統，是以 Photon 3.0 為基礎。為其他作業系統 (例如 Linux) 編譯的 Python 指令碼套件與執行階段容器不相容。當您嘗試將 Python 指令碼做為 vRealize Orchestrator 工作流程或動作的一部分時，此問題可能會導致 Python 指令碼失敗。在此情況下，記錄中會出現下列錯誤訊息：

```
-04:00errorCannot find module action
```

若要解決此問題，您必須在 Photon OS 容器資料夾中安裝必要的 Python 套件。

必要條件

安裝 Docker。請參閱[取得 Docker](#)。

程序

- 1 導覽至 Python 指令碼的父系資料夾。
- 2 將容器資料夾掛接至您的父系資料夾，以建立具有基礎 Photon 映像的容器。

**備註** 下列指令碼是單一 Docker 命令，必須完整執行才能建立適當的容器。

```
docker run -ti -v
$(pwd)/<name_of_folder_that_contains_your_python_script>/:/
<name_of_folder_that_contains_your_python_script>
python:3.0
```



**3** 在容器中安裝 Python。

```
tdnf install -y python3-3.7.5-5.ph3 python3-pip-3.7.5-5.ph3
```

**4** 導覽至包含 Python 指令碼的容器資料夾。

**5** 新增您的 Python 指令碼和套件。

---

**備註** 在 lib 資料夾中安裝您 Python 指令碼所需的套件。

---

```
pip3 install <package_name> -t lib/
```

**6** 退出容器，並導覽至您掛接至容器的本機資料夾。

**7** 將所有相關的檔案和資料夾壓縮為 ZIP 封存檔。

**8** 將 ZIP 封存檔匯入 vRealize Orchestrator 用戶端中，並將其做為動作的一部分執行以驗證指令碼。