使用 VMware vRealize Orchestrator 客户端

2022 年 2 月

vRealize Orchestrator 8.7

vmware®

您可以从 VMware 网站下载最新的技术文档：

https://docs.vmware.com/cn/。

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

威睿信息技术（中国）有限公司
北京办公室
北京市
朝阳区新源南路 8 号
启皓北京东塔 8 层 801
www.vmware.com/cn

上海办公室
上海市
淮海中路 333 号
瑞安大厦 804-809 室
www.vmware.com/cn

广州办公室
广州市
天河路 385 号
太古汇一座 3502 室
www.vmware.com/cn

目录

使用 VMware vRealize Orchestrator 客户端

1

《使用 VMware vRealize Orchestrator 客户端》提供了有关 vRealize Orchestrator Client 的工作流自动化特性和功能的信息。

目标读者

这些信息适用于经验丰富的系统管理员，其中介绍了有助于其运行和管理 vRealize Orchestrator 工作流的实用工具。

vRealize Orchestrator Client 简介 2

可使用 vRealize Orchestrator Client 管理 vRealize Orchestrator 服务和对象。

可以通过 `https://your_orchestrator_FQDN/orchestration-ui` 访问 vRealize Orchestrator Client。

UI 元素	说明
仪表板	使用 vRealize Orchestrator Client 仪表板和分析功能收集有关 vRealize Orchestrator 环境和工作流的有用衡量指标数据。
工作流	创建、编辑、调度、运行和删除工作流。
操作	创建、编辑和删除操作。操作编辑器支持自动完成 vRealize Orchestrator API Explorer 中包含的常见脚本元素。
策略	创建、编辑、运行和删除策略。
软件包	创建、删除、导出和导入包含 vRealize Orchestrator 对象的软件包。
配置	创建、运行和删除配置元素。
资源	导出、导入和更新资源元素。
组	具有管理员权限的用户可以将角色分配给 vRealize Orchestrator Client 中的用户，并将其添加到组中。
审核日志	查看在 vRealize Orchestrator Client 中记录的不同事件，例如在创建对象时记录的事件。
Git 存储库	创建与 Git 存储库的集成，并使用集成管理跨多个部署的工作流和其他 vRealize Orchestrator 对象的开发。 请参见如何使用 Git 分支管理我的 vRealize Orchestrator 对象清单。
已删除的项目	还原已删除的 vRealize Orchestrator Client 对象，例如工作流、操作、策略、配置元素和资源元素。
API Explorer	浏览 vRealize Orchestrator Client 中可用的 API 命令。 注 vRealize Orchestrator Client 通过 REST 代理与 vRealize Orchestrator REST API 进行通信。

本章讨论了以下主题：

- vRealize Orchestrator 客户端使用情况仪表板
- vRealize Orchestrator Client 中的内容组织

vRealize Orchestrator 客户端使用情况仪表板

vRealize Orchestrator Client 仪表板提供了一个有用的工具，可用于对 vRealize Orchestrator Client 工作流进行监控、管理和故障排除。

vRealize Orchestrator Client 仪表板上的信息分布在五个面板上。

窗口	描述
工作流运行	提供有关正在运行、正在等待和失败的工作流运行数的直观数据。
收藏的工作流	显示已添加到收藏夹的工作流。
等待输入	显示需要进一步用户交互的待处理工作流运行。这些工作流还会显示在用户界面右上角的通知菜单中。
最近的工作流运行	管理最近的工作流运行。显示工作流运行的名称、状态、开始日期和结束日期。
需要注意	显示失败的工作流运行和工作流运行性能衡量指标。

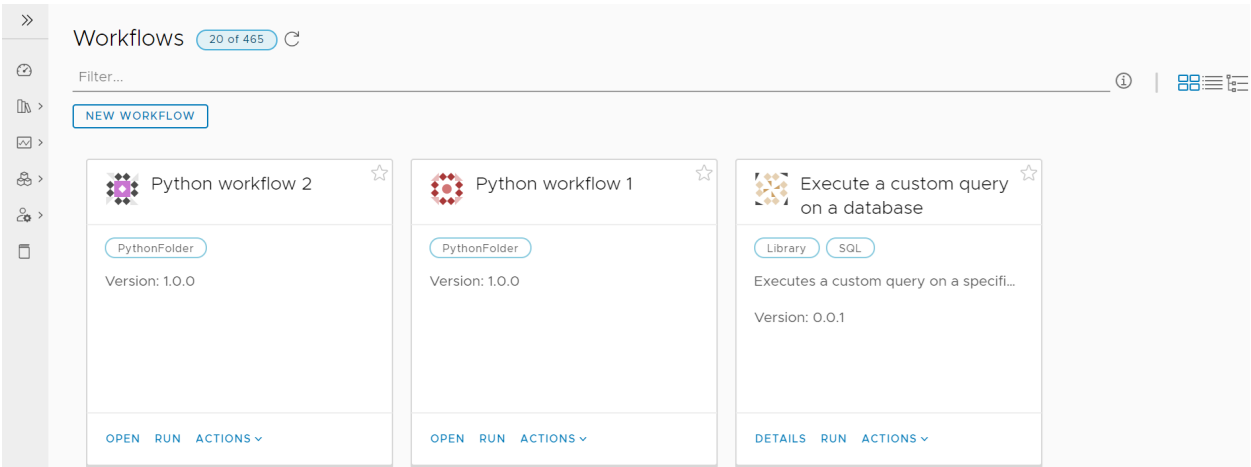
vRealize Orchestrator Client 中的内容组织

管理 vRealize Orchestrator 对象清单在 vRealize Orchestrator Client 中的显示方式。

vRealize Orchestrator Client 为工作流、操作、策略、资源和配置等对象提供三种不同类型的视图支持：卡视图、列表视图和树视图。您可以从页面右上角更改当前的视图类型。

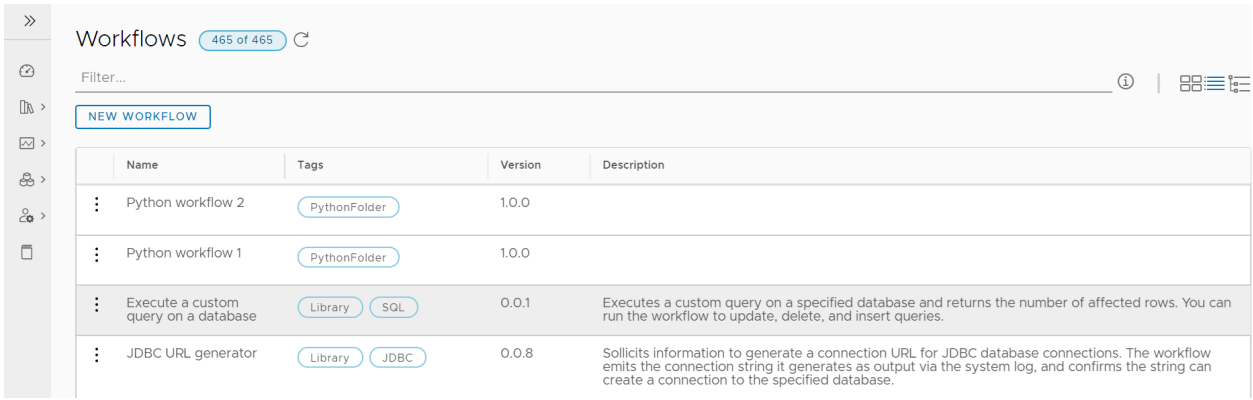
卡视图

卡视图是 vRealize Orchestrator Client 中使用的默认视图类型。各个清单对象的信息（如工作流）将显示在单独的卡视图元素中。



列表视图

列表视图以列表形式显示 vRealize Orchestrator 对象的信息。有关可在对象上执行的操作的详细信息，请单击对象左侧的垂直省略号图标。



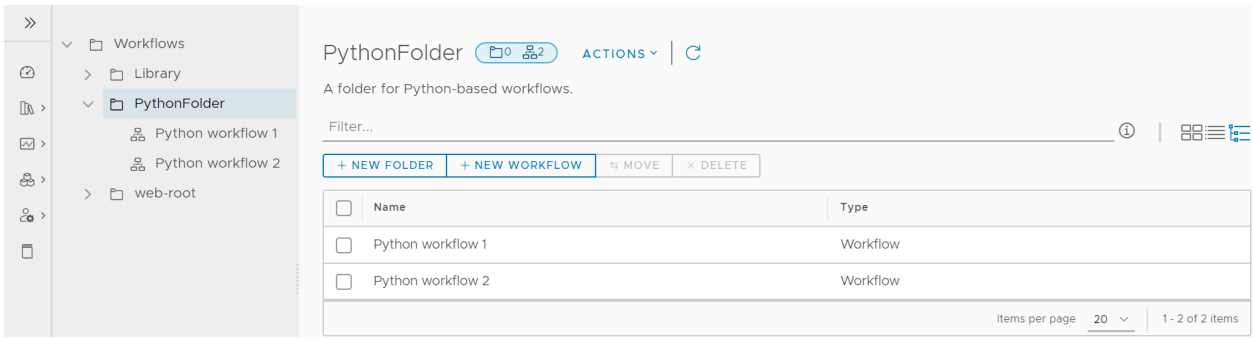
树视图

您可以在树视图中的分层文件夹下组织对象清单。每个 vRealize Orchestrator 对象类型都有一个根级别文件夹。您无法在根文件夹下创建新对象，如工作流。您必须创建在根文件夹下组织的单独文件夹。每个文件夹都包含可帮助您管理其内容的工具，如内容筛选器。

注 每个文件夹都有一个单独的内容筛选器。无法跨文件夹筛选内容。

有关文件夹的详细信息，请参见在 vRealize Orchestrator 客户端中创建文件夹或子文件夹。

注 从树视图中选择对象时，将以只读模式打开该对象。要编辑对象内容（如工作流变量或工作流结构定义），请单击顶部选项菜单中的**编辑**。



在 vRealize Orchestrator 客户端中创建文件夹或子文件夹

使用分层文件夹结构组织 vRealize Orchestrator 对象。

您可以创建文件夹和子文件夹来组织以下类型的 vRealize Orchestrator 对象：

- 工作流
- 操作

- 策略
- 配置元素
- 资源元素

步骤

1 登录到 vRealize Orchestrator Client。

2 从左侧导航窗格中，选择一个对象页面，如**工作流**。

3 选择右上角的树视图图标 ( )。

4 （可选） 要创建子文件夹，请从左侧的树视图中选择一个父文件夹。

5 单击**新建文件夹**。

6 输入名称和说明，然后单击**保存**。

7 向新创建的文件夹添加对象或子文件夹。

8 （可选） 要编辑文件夹名称，请选择**操作 > 编辑**。

在 vRealize Orchestrator Client 中移动对象和文件夹

通过将内容移动到另一个文件夹来重新组织您的 vRealize Orchestrator 内容。

无法在操作模块之间移动操作，也无法将任何对象移到根文件夹。根文件夹包含主对象文件夹和子文件夹，但不能用于存储对象。

步骤

1 登录到 vRealize Orchestrator Client。

2 从左侧导航窗格中，选择一个对象页面，如**工作流**。

3 选择右上角的树视图图标 ( )。

4 展开树视图，然后选择要移动的对象或文件夹。

5 将对象或文件夹拖到新的父文件夹中。

> 注 还可以直接从对象编辑器中将对象移动到新文件夹。在**摘要**选项卡上，单击**选择文件夹**，然后为该对象选择新的父文件夹。另一个移动选项是从文件夹页面上的表中选择对象。此选项对于执行包含多个 vRealize Orchestrator 对象的批处理移动操作非常有用。

删除 vRealize Orchestrator Client 中的文件夹或子文件夹

从 vRealize Orchestrator Client 中删除废弃的文件夹或子文件夹。

您无法删除每个 vRealize Orchestrator 对象类型对应的根级别文件夹。

步骤

1 登录到 vRealize Orchestrator Client。

2 从左侧导航窗格中，选择一个对象页面，如**工作流**。

3 选择右上角的树视图图标 ()。

4 勾选要删除的文件夹旁边的复选框。

注 要删除子文件夹，请从树视图中选择父文件夹，然后勾选复选框。

5 单击**删除**。

6 如果选定文件夹为空。

a 确认删除文件夹。

b 单击**删除**。

7 如果选定文件夹包含 vRealize Orchestrator Client 对象或子文件夹。

a 确认删除文件夹。

b 单击**删除**。

您将收到消息无法删除项目“`your_folder_name`”：文件夹“`your_folder_name`”不为空 (`Could not delete item 'your_folder_name': Folder 'your_folder_name' is not empty`)。

c 要删除该文件夹及其所有内容，请单击**强制删除**。

d 确认删除文件夹，然后单击**删除**。

注 还可以通过从文件夹菜单中包含的表中选择多个对象来执行批量删除操作。

设置 vRealize Orchestrator Client 3

要充分利用 vRealize Orchestrator Client 的功能，必须配置用户权限，并了解如何使用版本历史记录管理对象。

本章讨论了以下主题：

- vRealize Orchestrator 角色和组
- vRealize Orchestrator 对象版本历史记录

vRealize Orchestrator 角色和组

vRealize Orchestrator 管理员可以通过设置权限来控制对 vRealize Orchestrator Client 中的功能和内容的访问权限。访问权限分为用户角色和组权限。

角色用于控制用户可查看和使用的 vRealize Orchestrator Client 功能。对角色管理功能的访问权限取决于 vRealize Orchestrator 环境的许可证类型。

表 3-1. 对 vRealize Orchestrator 角色管理功能的基于许可证的访问权限

	身份验证	
许可证	**vSphere**	**vRealize Automation**
vSphere	不支持角色管理。组仅支持“运行”权限。	
vRealize Automation	在 vRealize Orchestrator Client 中管理角色。 请参见在 vRealize Orchestrator Client 中分配角色。	通过 vRealize Automation 中的**身份和访问管理设置**来管理角色。 请参见在 vRealize Automation 中配置 vRealize Orchestrator 客户端角色。

组权限用于控制用户可查看和使用的 vRealize Orchestrator Client 内容，如工作流、操作、策略、配置元素和资源元素。除非通过组权限另行配置，否则对标准工作流和操作等预配置的系统 vRealize Orchestrator 内容的访问权限将在所有用户之间共享。

对于具有管理员和查看者角色的用户，其访问权限不受组权限的限制。对于未分配任何角色的用户以及具有工作流设计人员角色的用户，其访问权限取决于为其分配的组。您可以通过修改这些用户所在组的权限来扩展其访问权限。通过这种方式，您可以将用户组织到公共项目中。例如，您可以创建一个组，然后将正在从事开发自定义 vRealize Orchestrator 插件工作的用户包含到该组中，并仅允许这些用户修改特定于其组的内容。

表 3-2. vRealize Orchestrator 用户角色和组权限

角色	访问权限		
管理员	管理员可以访问所有 vRealize Orchestrator Client 功能和内容，包括由特定组创建的内容。负责设置用户角色，创建和删除组以及将用户添加到组中。管理员不受组权限限制。 注 默认情况下，用于对 vRealize Orchestrator 进行身份验证的 vRealize Automation 环境中的租户管理员具有**管理员**权限。		
查看者	查看者对 vRealize Orchestrator Client 中的所有内容具有只读访问权限，但不能创建、编辑、运行或导出内容。查看者还可以查看所有组和组内容。查看者不受组权限限制。		
	组权限		
	未分配任何组	**运行**	**运行和编辑**
工作流设计器	■ 查看系统内容。 ■ 查看并运行自己的运行。 ■ 创建、运行、编辑和删除自己的内容。	■ 查看系统内容 ■ 查看并运行自己的运行。 ■ 创建、运行、编辑和删除自己的内容。 ■ 将自己的内容添加到组。 ■ 运行组内容，但不能对其进行编辑。	■ 查看系统内容。 ■ 查看并运行自己的运行。 ■ 创建、运行、编辑和删除自己的内容。 ■ 将自己的内容添加到组。 ■ 运行和编辑组内容。 注 不适用于通过 vSphere 进行身份验证的 vRealize Orchestrator 实例。
未分配任何角色的用户	■ 查看自己的运行。	■ 查看并运行自己的运行。 ■ 查看和运行组内容。	■ 查看并运行自己的运行。 ■ 查看和运行组内容。 注 要使此组中的用户能够创建、编辑和添加内容，必须为其分配工作流设计人员角色。 注 不适用于通过 vSphere 进行身份验证的 vRealize Orchestrator 实例。

在 vRealize Orchestrator Client 中分配角色

作为管理员，您可以将用户添加到 vRealize Orchestrator Client，并设置这些用户可以查看和使用的功能。

角色管理控制 vRealize Orchestrator 身份提供程序中的用户对 vRealize Orchestrator Client 功能的访问。角色管理同时涵盖 vRealize Orchestrator Client 用户界面和 API 功能。

注 客户端角色管理仅适用于通过 vSphere 进行身份验证并使用 vRealize Automation 许可证的 vRealize Orchestrator 实例。有关向使用 vRealize Automation 进行身份验证的 vRealize Orchestrator 分配角色的信息，请参见在 vRealize Automation 中配置 vRealize Orchestrator 客户端角色。

步骤

1 以管理员身份登录到 vRealize Orchestrator 客户端。

2 导航到**管理 > 角色管理**。

3 单击**添加**。

4 搜索要添加到 vRealize Orchestrator Client 中的用户或组。

5 选择用户的角色。有关角色的详细信息，请参见 vRealize Orchestrator 角色和组。

6 单击**保存**。

在 vRealize Automation 中配置 vRealize Orchestrator 客户端角色

可以在 vRealize Automation 的 **身份和访问管理**页面中为 vRealize Orchestrator Client 分配服务角色。可以为嵌入式 vRealize Orchestrator Client 和使用 vRealize Automation 进行了身份验证的独立 vRealize Orchestrator 实例分配服务角色。

vRealize Orchestrator 服务角色可以管理嵌入式 vRealize Orchestrator Client 用户可以访问的功能。有关 vRealize Orchestrator 角色的详细信息，请参见 vRealize Orchestrator 角色和组。

注 通过 vSphere 进行身份验证并使用 vRealize Automation 许可证的 vRealize Orchestrator 实例可以直接在 vRealize Orchestrator Client 中分配角色。请参见在 vRealize Orchestrator Client 中分配角色。

前提条件

- 确认已从有效的 vIDM 实例中导入适当的用户和组。
- 在向用户分配 vRealize Orchestrator 服务角色之前，请确认已向您的用户分配 vRealize Automation 中的组织角色。请参见《管理 vRealize Automation》中的管理 vRealize Automation 中的用户和组。

步骤

1 从右上角的标题下拉菜单中，选择**身份和访问管理**选项。

2 在**活动用户**选项卡上，搜索要分配给 vRealize Orchestrator 的用户的电子邮件地址。

3 选中用户旁边的复选框，然后单击**编辑角色**。

4 单击**添加服务访问**。

5 从左侧的下拉菜单中选择 **Orchestrator**。

6 从右侧的下拉菜单中选择要分配给该用户的角色。

7 单击**保存**。

在 vRealize Orchestrator Client 中创建组

作为管理员，您可以在 vRealize Orchestrator Client 中使用组来设置用户可查看和访问的 vRealize Orchestrator 内容。

可以使用 vRealize Orchestrator Client 将组权限设置为 vRealize Orchestrator 工作流、操作、策略、配置元素、资源元素和软件包。

注 对于使用 vSphere 进行身份验证的 vRealize Orchestrator 实例中的用户，只能拥有**运行**组权限。

步骤

1 以管理员身份登录到 vRealize Orchestrator 客户端。

2 导航到**管理 > 组**。

3 单击**新建组**。

4 在**摘要**选项卡中，添加组的名称和描述。

5 在**用户**选项卡中，单击**添加**。

   a 搜索要添加到组中的用户。

   b 为用户分配组权限。

   c 单击**添加**。

6 在**项目**选项卡中，将 vRealize Orchestrator 对象添加到组。

   注 在 vRealize Orchestrator Client 中创建对象时也可以将该对象添加到现有组。要添加对象，请从对象编辑器的**摘要/常规**选项卡上的**可访问者**下拉菜单中选择组。

7 单击**保存**。

vRealize Orchestrator 对象版本历史记录

vRealize Orchestrator Client 将保留每个 vRealize Orchestrator 对象的版本历史记录。使用版本历史记录，您可以比较不同的 vRealize Orchestrator 对象版本以及恢复到以前的版本。

保存每个 vRealize Orchestrator 对象时，vRealize Orchestrator 会创建该对象的版本历史记录。后续对 vRealize Orchestrator 对象进行更改时，将创建新的版本历史记录。以前的版本历史记录会保留，并且可用于跟踪对相关对象进行的更改以及将对象恢复到以前版本。将对象恢复到以前的版本会创建新版本的历史记录。

vRealize Orchestrator Client 跟踪以下 vRealize Orchestrator 对象的版本历史记录：

- 工作流
- 操作
- 软件包
- 策略

- 资源元素
- 配置元素

注 生成的工作流不会显示在工作流版本历史记录中。例如，由**为表格生成 CRUD 工作流**工作流生成的工作流不会显示在**版本历史记录**选项卡中，无法推送到已配置的任何 Git 存储库。要将生成的这些工作流包含到 vRealize Orchestrator 版本历史记录中，请复制这些工作流。

可以从对象编辑器页面的**版本历史记录**选项卡访问对象的版本历史记录。如果您尝试与其他用户同时编辑对象，则可能会发生合并冲突。要解决合并冲突，请单击错误消息右侧的**解决**。在**解决冲突**窗口中，有三个选项：

- **使用他们的**。使用其他用户所做的更改以解决合并冲突。
- **使用我们的**。使用您的更改以解决合并冲突。
- **解决**。通过编辑显示的更改模型解决合并冲突。如果提供的模型无效，则此选项不可用。

将工作流还原到先前版本

您可以将工作流还原到以前保存的版本。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 工作流**，然后选择工作流。

3 选择**版本历史记录**选项卡。

4 要查看各版本之间的比较，请选择一个工作流版本，然后从**差异对象**下拉菜单中选择另一个版本。

窗口会显示当前工作流版本与选定工作流版本之间的差异。

5 要将工作流还原到另一个版本，请单击**还原**。

工作流状态会恢复到选定版本的状态。

注 还可以通过图形差异工具视图还原工作流版本。请参见工作流版本之间的可视化比较。

工作流版本之间的可视化比较

使用图形差异工具比较各工作流版本之间的变更。

默认情况下，vRealize Orchestrator 版本历史记录会在一个 YAML 表单中显示各工作流版本之间的差异。您还可以在不同的工作流版本之间执行可视化比较。您可以查看以下方面的变更：

- 常规工作流信息，如版本号和工作流说明。
- 工作流中使用的变量。
- 工作流的输入和输出参数。
- 工作流结构定义。

前提条件

创建工作流。

步骤

1 登录到 vRealize Orchestrator Client。

2 导航到**库 > 工作流**，然后选择其中一个工作流。

3 编辑工作流的内容。

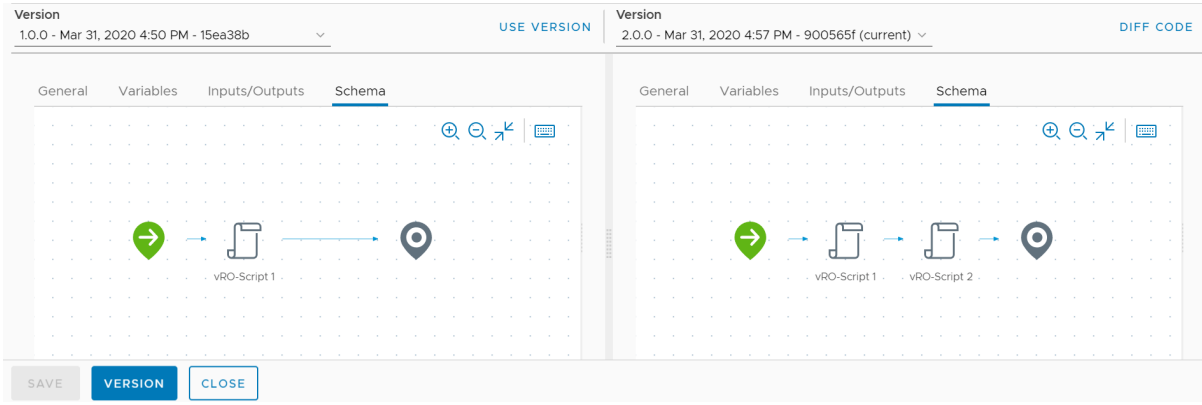
例如，可以在**结构定义**选项卡上添加额外的**可编辑脚本的任务**元素。

4 单击**保存**。

5 选择**版本历史记录**选项卡。

6 选择右上角的**视觉差异**。

现在，您可以在选择的两个工作流版本之间执行可视化比较。您可以从**版本**下拉菜单中选择要比较的版本。



7 （可选） 您可以通过选择**使用版本**将工作流还原到其他版本。

使用 Git 将 vRealize Orchestrator 内容清单重置为以前的状态

通过使用先前的 Git commit，可以将 vRealize Orchestrator 内容重置为先前的状态。

可以通过选择特定 commit 将 vRealize Orchestrator 内容重置为以前的状态。

前提条件

- 配置与 GitHub 或 GitLab 存储库的连接。请参见配置与 Git 存储库的连接。
- 将本地修改集推送到配置的 Git 存储库。

步骤

1 登录到 vRealize Orchestrator Client。

2 导航到**管理 > Git 历史记录**。

3 选择要重置为的修改集，然后单击**重置为此设置**。

4 确认要重置为此特定 commit，然后单击**确定**。

vRealize Orchestrator 内容清单将重置为此 commit 中指定的状态。相关的 vRealize Orchestrator 内容将恢复到以前的版本。如果推送 commit 时内容不存在，则会将其从清单中移除。

后续步骤

要将 vRealize Orchestrator 清单还原到保存到 Git 存储库的最新状态，请从 **Git 历史记录**窗口执行 `Pull` 命令。

vRealize Orchestrator 用例

4

这些用例演示了 vRealize Orchestrator 平台的部分功能。

这些用例仅提供示例值。您自己的环境结构和命名约定可能不同。

本章讨论了以下主题：

- 如何在 vRealize Orchestrator 中使用 Python 集成 Amazon Web Services
- 如何使用 Git 分支管理我的 vRealize Orchestrator 对象清单
- 如何使用第三方模块调用 vRealize Automation 项目 API

如何在 vRealize Orchestrator 中使用 Python 集成 Amazon Web Services

此 vRealize Orchestrator 用例显示了如何使用 Python 扩展 vRealize Orchestrator 部署功能的示例。

您可以在操作和工作流脚本中使用以下运行时：

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

> **注** PowerCLI 运行时包括 PowerShell 和以下模块：VMware.PowerCLI、PowerNSX 和 PowervRA。

重要事项 仅当您的 vRealize Orchestrator 部署使用 vRealize Automation 许可证时，才能使用新运行时。

此用例演示了如何创建在 Amazon Web Services (AWS) 中调用 EC2 实例的 Python 脚本。

重要事项 开始开发自定义脚本之前，确认您熟悉在 vRealize Orchestrator 中使用 Python、Node.js 和 PowerShell 脚本的核心概念。请参见 Python、Node.js 和 PowerShell 脚本的核心概念 。

步骤

1 创建初始 Python 脚本

在本地计算机上，创建 Python 脚本并将该脚本和 boto3 库打包为 ZIP 文件夹。

2 创建 Amazon Web Services 操作

创建使用 Python 脚本的 vRealize Orchestrator 操作。

3 调试 Amazon Web Services 操作

Python 脚本的原始版本故意设置了一个内置错误，以便于您学习如何调试脚本。

4 更新 Amazon Web Services 操作

导入更新的 Python 脚本，然后再次运行操作。

创建初始 Python 脚本

在本地计算机上，创建 Python 脚本并将该脚本和 boto3 库打包为 ZIP 文件夹。

前提条件

- 下载并安装 Python 3。请参见 Python 下载页面。
- 下载并安装 Visual Studio Code。请参见 Visual Studio Code 下载页面。
- 确认您已安装适用于 Visual Studio Code 的 Python 扩展。请参见 Visual Studio Marketplace。

步骤

1 在本地计算机上，创建一个 `vro-python-aws` 文件夹，然后在该文件夹上安装 boto3 Python SDK。

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

2 打开编辑器，并创建主要 Python 脚本。在此用例中，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 脚本列出给定区域中正在运行的所有 EC2 实例。

3 将创建的脚本以 `main.py` 文件的形式保存在 `vro-python-aws` 文件夹中。

4 登录到命令行界面。

5 导航至 `vro-python-aws` 文件夹。

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

6 创建一个包含 Python 脚本的 ZIP 包。

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

注 您也可以使用 ZIP 实用程序工具（如 7-Zip）创建 ZIP 包。

结果

您已创建基础 Python 脚本，并准备将其导入到 vRealize Orchestrator 部署中。

创建 Amazon Web Services 操作

创建使用 Python 脚本的 vRealize Orchestrator 操作。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 操作**。

3 单击**新建操作**。

4 在**常规**选项卡上，输入操作的名称、模块和版本号。

5 在**脚本**选项卡上，选择 **Python 3.7** 作为运行时，并选择 **Zip** 作为脚本类型。

6 单击**导入**。

7 浏览到 `vro-python-aws` 文件夹，然后选择包含 Python 脚本的 ZIP 包。

8 在**条目处理程序**文本框中，输入 **`main.handler`**。

注 操作的条目处理程序基于导入的 ZIP 包中的主脚本。由于主脚本位于 `main.py` 文件以及 **`handler`** 函数中，因此条目处理程序必须是 **`main.handler`**。如果您为主脚本文件提供了不同的标题，请相应地更改条目处理程序值。

9 保存操作，然后单击**运行**。

操作运行会遇到错误。

10 选择**日志**选项卡。

操作运行的日志显示 `"botocore.exceptions.NoRegionError: You must specify a region."` 错误消息。这是预期行为，因为初始 Python 脚本未定义区域。

后续步骤

调试 Python 脚本。请参见调试 Amazon Web Services 操作。

调试 Amazon Web Services 操作

Python 脚本的原始版本故意设置了一个内置错误，以便于您学习如何调试脚本。

前提条件

登录到您的 Amazon Web Services (AWS) 帐户，并专门为此用例场景创建一个 IAM 用户。请参见在 AWS 帐户中创建 IAM 用户。该 IAM 用户必须具有以下权限：

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

步骤

1 准备 vRealize Orchestrator Appliance。

小心 请勿在生产 vRealize Orchestrator 部署中调试脚本。从用于开发和测试的单节点 vRealize Orchestrator 部署中进行调试。

a 以 **root** 用户身份通过 SSH 登录到 vRealize Orchestrator Appliance 命令行。

b 运行 `vracli dev tools` 命令。

c 系统会提示确认是否要继续。输入 **yes** 继续；输入 **no** 取消。

重要事项 通过运行 `vracli dev tools` 命令，打开调试 Python 脚本所需的端口。在调试过程中，必须确保当前 SSH 会话处于打开状态。

2 启动调试配置。

a 登录到 vRealize Orchestrator 客户端。

b 打开 AWS 操作，然后单击**调试**。

调试过程将开始，并且操作运行将暂停。

c 选择**调试配置**选项卡。

该选项卡包含一个 `.json` 配置，您可以将其远程连接到 IDE 以调试 Python 脚本。

d 手动复制配置内容，或单击**复制到剪贴板**。

3 调试 Python 脚本。

a 打开 Visual Studio Code。

b 打开 `vro-python-aws` 文件夹。

c 从顶部的导航窗格中，选择**运行 > 打开的配置**。

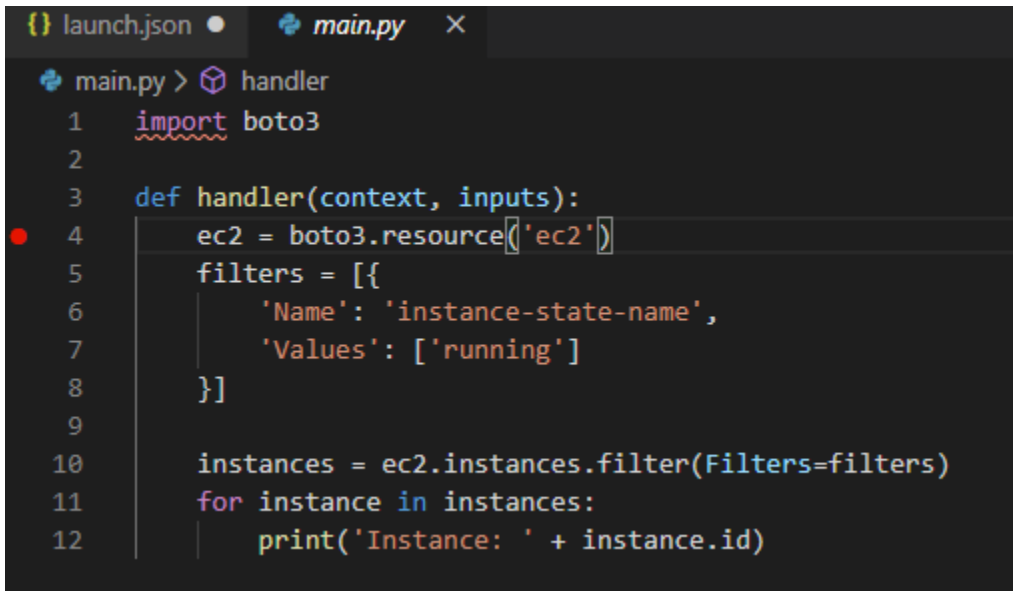
d 选择 **Python 文件**。

e 将 `"version"` 和 `"configuration"` 属性保留在当前位置，然后粘贴从 vRealize Orchestrator 客户端复制的 `.json` 配置的内容。生成的 `launch.json` 文件必须如下所示：

```
{

    "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 选择 `main.py` 脚本文件，然后将断点添加到 `ec2 = boto3.resource('ec2')` 行。



g 从顶部的导航窗格中，选择**运行 > 启动调试**。

h 当调试器到达断点时，执行单步跳过操作。

调试运行指示 Python 脚本缺少指定的区域和 AWS 访问密钥。

i 返回打开的 vRealize Orchestrator Appliance 会话，然后按 **Enter** 关闭为此调试会话打开的端口。

4 将缺少的信息添加到 Python 脚本。

a 在 Visual Studio Code 中，创建一个 `awsconfig` 文件，其中包含 IAM 用户的 AWS 访问密钥以及要使用 Python 脚本进行 ping 的 AWS 区域。

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

b 将 `awsconfig` 另存为 `vro-python-aws` 文件夹中的配置 (.cfg) 文件。

c 打开 `main.py` 文件并对其进行修改，以便 boto3 库可以使用 `awsconfig.cfg` 文件。

```
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 创建一个新的 ZIP 包，其中包含 `main.py` 文件、`awsconfig.cfg` 文件和 boto3 库。

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

注 您也可以使用 ZIP 实用程序工具（如 7-Zip）创建 ZIP 包。

更新 Amazon Web Services 操作

导入更新的 Python 脚本，然后再次运行操作。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 操作**，然后选择原始 Amazon Web Services (AWS) 操作。

3 （可选）在**常规**选项卡上，更改版本号。

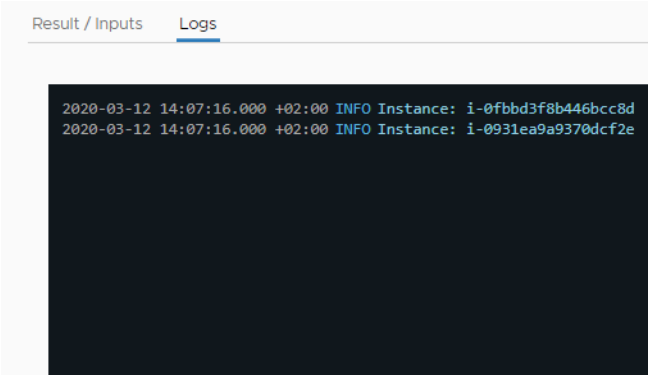
4 移除旧 ZIP 包，然后单击**导入**。

5 选择更新的 ZIP 包。

6 保存操作，然后单击**运行**。

7 操作运行完成后，选择**日志**选项卡。

日志将显示操作查询的 EC2 实例。



后续步骤

创建一个 vRealize Orchestrator 工作流，该工作流使用更新的 AWS 操作作为**操作元素**。

如何使用 Git 分支管理我的 vRealize Orchestrator 对象清单

使用分支组织如何在 Git 存储库中管理 vRealize Orchestrator 内容。

通过使用 Git，可以提供集中式存储库，从而提高 vRealize Orchestrator 开发人员的灵活性。例如，可以使用 Git 跨多个 vRealize Orchestrator 环境管理工作流开发。

注 要使用 Git 管理对象清单，vRealize Orchestrator 部署必须使用 vRealize Automation 许可证。有关详细信息，请参见《安装和配置 vRealize Orchestrator》中的“使用许可证启用 vRealize Orchestrator 功能”。

现在，您可以将对象推送到分支以及从分支中提取对象。您可以使用分支管理特定 vRealize Orchestrator 对象组的开发，然后再将这些对象重新合并到主分支中。

在此用例中，使用 GitLab 项目来管理使用 Python 运行时的 vRealize Orchestrator 对象。此用例表示 vRealize Orchestrator 中的 Git 功能示例，不表示功能范围限制。

注 如果您更熟悉 GitHub，则可以在此用例中使用 GitHub 存储库。

步骤

1 准备 GitLab 环境

为 vRealize Orchestrator Python 对象创建 Git 分支。

2 配置与 Git 存储库的连接

作为**管理员**，您可以在 vRealize Orchestrator 部署和 Git 存储库或项目之间配置连接。

3 将修改推送到 Git 存储库

将对本地 vRealize Orchestrator 对象的更改推送到集成 Git 存储库。在此用例中，我们将对基于 Python 的 vRealize Orchestrator 操作的更改推送到特定 Git 分支。

准备 GitLab 环境

为 vRealize Orchestrator Python 对象创建 Git 分支。

前提条件

为 vRealize Orchestrator 环境创建 GitLab 项目。请参见创建项目。

步骤

1 登录到您的 GitLab 帐户。

2 导航到您的 GitLab 项目。

3 在左侧导航窗格中，选择**存储库 > 分支**。

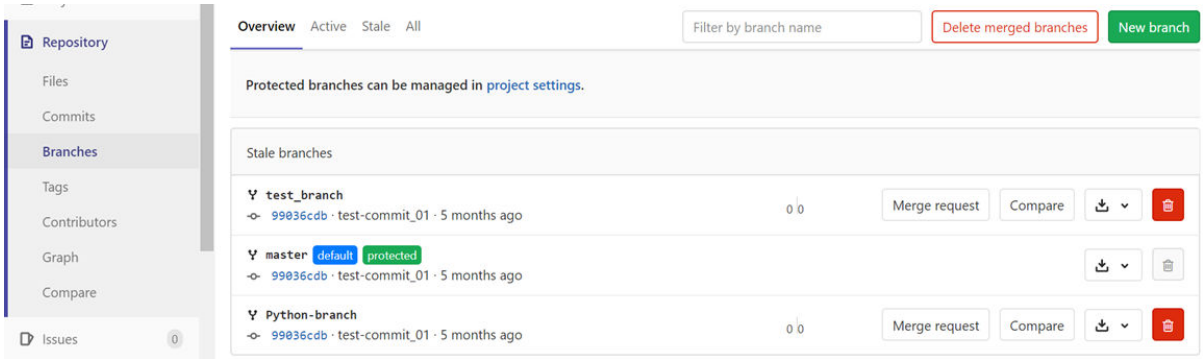
4 在**概览**选项卡上，单击**新建分支**。

5 在**分支名称**下，输入 **`Python-Branch`**。

6 将**创建自**选项作为**主分支**。

7 单击**创建分支**。

您已为基于 Python 的 vRealize Orchestrator 对象创建一个分支。



配置与 Git 存储库的连接

作为**管理员**，您可以在 vRealize Orchestrator 部署和 Git 存储库或项目之间配置连接。

要使用 Git 管理 vRealize Orchestrator 对象清单，必须使用 vRealize Orchestrator Client 配置与 Git 存储库的连接。

注 您无法通过 SSH 添加来自不同帐户的多个 Git 存储库，因为 vRealize Orchestrator 会为每个实例各创建一个 SSH 密钥。要添加多个 Git 存储库，您可以按照本文档中所述通过 HTTP 添加这些存储库。

前提条件

- 确认 vRealize Orchestrator 环境使用 vRealize Automation 许可证。

- 为 GitLab 项目生成访问令牌并将其复制到剪贴板，以便在配置过程中使用。请参见创建个人访问令牌。

  注 在此用例中，您使用的是 GitLab 项目。如果您更熟悉 GitHub，则可以使用 GitHub 存储库。有关生成 GitHub 令牌的信息，请参见为命令行创建个人访问令牌。

步骤

1 以**管理员**身份登录到 vRealize Orchestrator Client。

2 导航到**管理 > Git 存储库**。

3 单击**添加存储库**。

4 输入 Git 存储库的 URL 地址。

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

  注 还可以使用 SSH 协议建立连接。

5 输入 Git 配置文件的用户名。

6 输入 Git 存储库的访问令牌。

7 要验证与 Git 存储库的连接，请单击**验证**。

8 （可选） 更改用于在 vRealize Orchestrator Client 中标识存储库的名称。

9 （可选） 为连接的 Git 存储库添加简短描述。

10 要激活连接的 Git 存储库，请单击**创建活动存储库**。

  注 一次只能有一个 Git 存储库处于活动状态。您可以从 **Git 存储库**页面更改活动 Git 存储库。

11 选择要推送更改的分支。在此用例中，您使用的是 **Python-branch**。请参见准备 GitLab 环境。

  注 完成初始 Git 配置后，您可以随时更改选择的 Git 分支。

12 要完成配置过程，请单击**保存**。

后续步骤

导航返回至 **Git 存储库**菜单，并确认存储库的状态为**活动**。

将修改推送到 Git 存储库

将对本地 vRealize Orchestrator 对象的更改推送到集成 Git 存储库。在此用例中，我们将对基于 Python 的 vRealize Orchestrator 操作的更改推送到特定 Git 分支。

您可以将本地更改集推送到 Git 存储库。每个更改集可以包含一个或多个修改后的 vRealize Orchestrator 对象。

注 将更改集推送到 Git 存储库和放弃更改集的过程不受组权限的限制。因此，一个组中的工作流开发人员可以推送或放弃其他开发人员所做的本地更改。

前提条件

- 验证是否已创建 Git 分支。请参见准备 GitLab 环境。
- 验证是否已配置与 Git 存储库的连接。请参见配置与 Git 存储库的连接。
- 验证是否已将您的 Git 集成设置为将更改推送到 **Python-branch** Git 分支。
- 创建基于 Python 的 vRealize Orchestrator 对象。例如，请参见如何在 vRealize Orchestrator 中使用 Python 集成 Amazon Web Services。

步骤

1 登录到 vRealize Orchestrator Client。

2 编辑您的 Python 操作。

   a 导航到**库 > 操作**，然后选择 Python 操作。

   b 对操作进行一些小的更改，例如更改说明。

   c 保存操作。

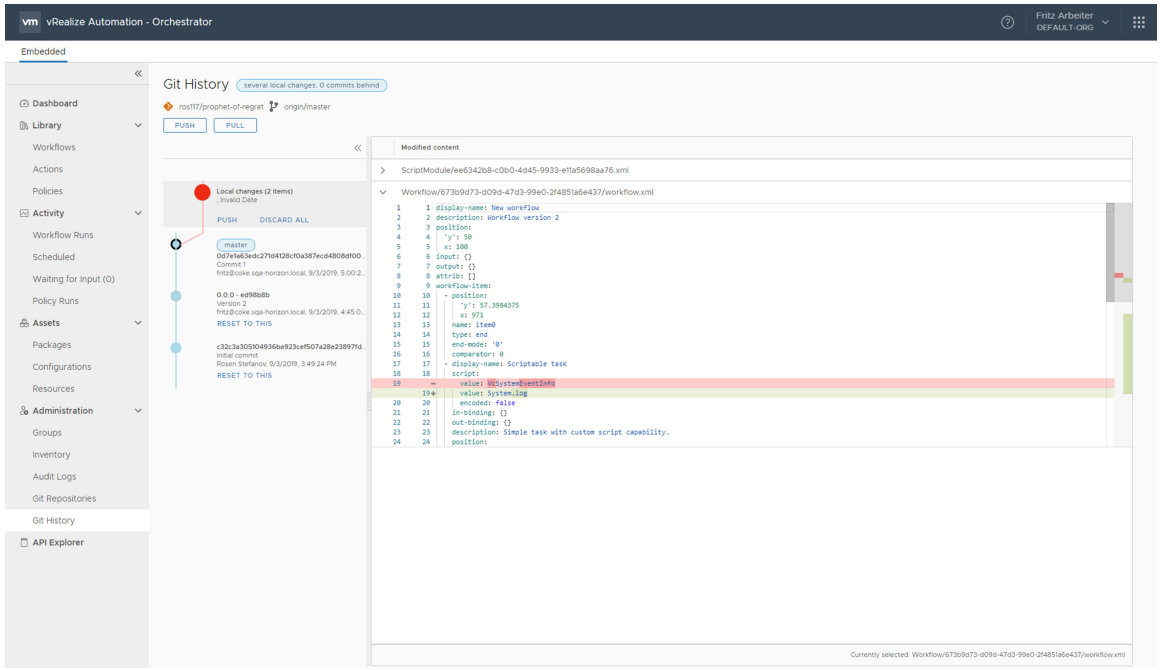
3 将所做更改推送到 Git 存储库。

> **注** 您也可以通过单击对象编辑器底部显示的**版本**选项，在对象级别推送本地修改。

a 导航到**管理 > Git 历史记录**。

Git 历史记录显示本地版本分支与选定 Git 存储库分支之间的当前差异。您可以展开任何修改后的 vRealize Orchestrator 对象的条目查看版本差异。

> **注** 可以通过选择**全部放弃**放弃本地更改集。



b 单击**推送**。

c 输入 commit 标题。

d （可选） 输入 commit 的简短描述。

e 选择要推送到 Git 存储库的 Python 操作更改。

4 要将本地更改集推送到 Git 存储库，请单击**推送**。

后续步骤

完成 Git 分支的开发工作后，将该分支与主分支合并。请参见如何创建合并请求。

如何使用第三方模块调用 vRealize Automation 项目 API

此 vRealize Orchestrator 用例说明了如何使用第三方模块调用 vRealize Automation 项目 API。

您可以在操作和工作流脚本中使用以下运行时：

- Python 3.7

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

> **注** PowerCLI 运行时包括 PowerShell 和以下模块：VMware.PowerCLI、PowerNSX 和 PowervRA。

在此用例中，您将了解如何创建使用第三方依赖关系模块连接到 vRealize Automation 项目 API 的 vRealize Orchestrator 操作。

重要事项 开始开发自定义脚本之前，确认您熟悉在 vRealize Orchestrator 中使用 Python、Node.js 和 PowerShell 脚本的核心概念。请参见 Python、Node.js 和 PowerShell 脚本的核心概念 。

创建调用 vRealize Automation 项目 API 的 Python 脚本

创建使用 Python 调用 vRealize Automation 项目 API 的示例脚本。

前提条件

确认您已安装 Python 3 和 PIP 软件包安装程序。请参见 Python 下载页面和 Python 软件包索引。

步骤

1 在本地计算机上，打开命令行 shell。

2 创建 `vro-python-vra` 文件夹。

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

3 导航到 `vro-python-vra` 文件夹。

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

4 创建名为 `handler.py` 的 Python 脚本。

```
touch handler.py
```

`handler.py` 脚本必须定义一个接受以下两个参数的函数：vRealize Orchestrator 工作流运行的上下文和绑定的 vRealize Orchestrator 输入。

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

> **注** 使用标准日志记录库时，在使用脚本的操作中记录的所有内容也会显示在工作流日志中。脚本的输入和返回内容必须在 vRealize Orchestrator 客户端中具有已配置的相应输入参数和返回类型。例如，脚本中的 `vRAUrl` 输入必须在 vRealize Orchestrator 客户端中具有名为 `vRAUrl` 的相应输入参数。同样，如果脚本返回字符串值，则在 vRealize Orchestrator 客户端中配置的返回类型也必须是字符串类型。如果操作返回复合类型对象，则可以使用 `Properties` 或 `Composite Type` 返回类型。

5 安装 Python 请求模块。

重要事项 第三方依赖关系模块必须安装在主 `vro-python-vra` 脚本文件夹中的根级别文件夹中。在此用例中，您需要为请求模块创建一个 `lib` 文件夹。

a 创建 `lib` 文件夹。

```
mkdir lib
```

b 安装请求模块。

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

6 将请求模块添加到 `handler.py` 脚本中。

```
import requests

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

7 创建对 vRealize Automation 项目 API 的 GET 请求。

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


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

8 定义 `token` 和 `vRAUrl` 值。

a 使用 vRealize Automation 身份验证服务 API 检索访问令牌。请参见获取 vRealize Automation API 的访问令牌

b 对于 `vRAUrl` 值，请定义脚本，使其使用具有相同名称的 vRealize Orchestrator 输入参数。

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

c 将新值添加到 `handler.py` 文件中。

```
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()
```

注 由于来自 vRealize Automation 项目 API 的响应为 JSON 格式，因此请对 vRealize Orchestrator 操作使用 `Properties` 或 `Composite Type` 返回类型。

9 创建一个 ZIP 包，其中包含 `handler.py` 文件和请求模块的 `lib` 文件夹。

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

后续步骤

将 PowerShell 脚本导入到 vRealize Orchestrator 操作。请参见在 vRealize Orchestrator 客户端中创建操作。

创建调用 vRealize Automation 项目 API 的 Node.js 脚本

创建使用 Node.js 调用 vRealize Automation 项目 API 的示例脚本。

前提条件

下载并安装 Node.js 14。请参见 Node.js 下载。

步骤

1 在本地计算机上，打开命令行 shell。

2 创建 `vro-node-vra` 文件夹。

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

3 导航到 `vro-node-vra` 文件夹。

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

4 创建名为 `handler.js` 的 Node.js 脚本。

```
touch handler.js
```

`handler.js` 脚本必须定义一个接受如下两个参数的函数：vRealize Orchestrator 工作流运行的上下文和绑定的 vRealize Orchestrator 输入。

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

注 使用标准日志记录库时，在使用脚本的操作中记录的所有内容也会显示在工作流日志中。脚本的输入和返回内容必须在 vRealize Orchestrator 客户端中具有已配置的相应输入参数和返回类型。例如，脚本中的 `vRAUrl` 输入必须在 vRealize Orchestrator 客户端中具有名为 `vRAUrl` 的相应输入参数。同样，如果脚本返回字符串值，则在 vRealize Orchestrator 客户端中配置的返回类型也必须是字符串类型。如果操作返回复合类型对象，则可以使用 `Properties` 或 `Composite Type` 返回类型。

5 安装 Node.js 请求模块。

```
npm install request
```

重要事项 第三方依赖关系模块必须安装在主 `vro-node-vra` 脚本文件夹中的根级别 `node_modules` 文件夹中。请勿移动或重命名此文件夹。

6 将请求模块添加到 `handler.js` 脚本中。

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

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

7 创建对 vRealize Automation 项目 API 的 GET 请求。

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

8 定义 `token` 和 `vRAUrl` 值。

a 使用 vRealize Automation 身份验证服务 API 检索访问令牌。请参见获取 vRealize Automation API 的访问令牌。

b 对于 `vRAUrl` 值，请定义脚本，使其使用具有相同名称的 vRealize Orchestrator 输入参数。

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

c 将新值添加到 `handler.js` 文件中。

```
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));
    });
}
```

注 由于来自 vRealize Automation 项目 API 的响应为 JSON 格式，因此请对 vRealize Orchestrator 操作使用 `Properties` 或 `Composite Type` 返回类型。

9 创建一个 ZIP 包，其中包含 `handler.js` 文件和请求模块的 `node_modules` 文件夹。

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

后续步骤

将 Node.js 脚本导入到 vRealize Orchestrator 操作。请参见在 vRealize Orchestrator 客户端中创建操作。

创建调用 vRealize Automation 项目 API 的 PowerShell 脚本

创建使用 PowerShell 调用 vRealize Automation 项目 API 的示例脚本。

步骤

1 在本地计算机上，打开命令行 shell。

2 创建 `vro-powershell-vra` 文件夹。

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

3 导航到 `vro-powershell-vra` 文件夹。

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

4 创建名为 handler.ps1 的 PowerShell 脚本。

```
touch handler.ps1
```

handler.ps1 脚本必须定义一个接受以下两个参数的函数：vRealize Orchestrator 工作流运行的上下文和绑定的 vRealize Orchestrator 输入。

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


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

注 使用标准日志记录库时，在使用脚本的操作中记录的所有内容也会显示在工作流日志中。脚本的输入和返回内容必须在 vRealize Orchestrator 客户端中具有已配置的相应输入参数和返回类型。例如，脚本中的 vRAUrl 输入必须在 vRealize Orchestrator 客户端中具有名为 vRAUrl 的相应输入参数。同样，如果脚本返回字符串值，则在 vRealize Orchestrator 客户端中配置的返回类型也必须是字符串类型。如果操作返回复合类型对象，则可以使用 Properties 或 Composite Type 返回类型。

5 安装 PowerShell 断言模块。

重要事项 第三方依赖关系模块必须安装在主 vro-powershell-vra 脚本文件夹中的根级别文件夹中。在此用例中，请为您的断言模块创建一个 Modules 文件夹。

a 创建 Modules 文件夹。

```
mkdir Modules
```

b 安装断言模块。

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

6 将断言模块添加到 handler.ps1 脚本中。

```
Import-Module Assert

function Handler {
  Param($context, $inputs)


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

7 创建对使用 Invoke-RestMethod cmdlet 的 vRealize Automation 项目 API 的 GET 请求。

```
$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 定义 `token` 和 `vRAUrl` 值。

a 使用 vRealize Automation 身份验证服务 API 检索访问令牌。请参见获取 vRealize Automation API 的访问令牌。

b 添加 `Assert-NotNull` 和 `Assert-Type` 断言模块属性。

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

c 对于 `vRAUrl` 值，请定义脚本，使其使用具有相同名称的 vRealize Orchestrator 输入参数。

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

d 将新值添加到 `handler.ps1` 文件中。

```
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
}
```

注 由于来自 vRealize Automation 项目 API 的响应为 JSON 格式，因此请对 vRealize Orchestrator 操作使用 `Properties` 或 `Composite Type` 返回类型。

9 创建一个 ZIP 包，其中包含 `handler.ps1` 文件和断言模块的 `Modules` 文件夹。

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

后续步骤

将 PowerShell 脚本导入到 vRealize Orchestrator 操作。请参见在 vRealize Orchestrator 客户端中创建操作。

管理工作流

5

工作流是按顺序运行的一组操作和决策。vRealize Orchestrator 提供了执行常见管理任务的工作流库。vRealize Orchestrator 还提供了工作流执行的各个操作的库。

工作流可以将多种操作、决策和结果整合在一起，按照特定顺序执行，从而在虚拟环境中完成特定的任务或进程。工作流可执行的任务包括虚拟机置备、备份、定期维护、发送邮件、执行 SSH 操作、管理物理基础架构和其他常规实用操作。工作流可按其功能接受输入值。您可以创建可按规定时间运行或在特定预期事件发生时运行的工作流。相关信息可由您、其他用户、其他工作流或操作或外部进程（例如由应用程序发出的 Web 服务调用）来提供。工作流可在运行前对这些信息进行验证和筛选。

工作流还可以调用其他工作流。例如，可以让工作流调用另一个工作流来创建新的虚拟机。

您可以使用 vRealize Orchestrator Client 界面的集成开发环境 (IDE) 创建工作流，通过 IDE，可以访问工作流库，还能够在工作流引擎上运行工作流。工作流引擎也可以从插入到 vRealize Orchestrator 中的外部库获取对象。此功能可以帮助您自定义流程或执行第三方应用程序提供的功能。

本章讨论了以下主题：

vRealize Orchestrator 工作流库中的标准工作流

vRealize Orchestrator 提供了一个标准的工作流库，您可以使用这些工作流在虚拟基础架构中自动执行相关操作。标准库中的工作流锁定为只读状态。要自定义标准工作流，必须复制该工作流。创建的副本工作流或自定义工作流均可进行完全编辑。

工作流库的内容可通过基于 HTML5 的 vRealize Orchestrator Client 的**库 > 工作流**菜单进行访问。客户端中的标准工作流和自定义工作流都使用标记进行组织。例如，您可以通过在工作流库搜索框中输入 **SSH** 来访问**生成密钥对**工作流。

注　只有复制工作流才能将新标记添加到标准工作流中。

在 vRealize Orchestrator 客户端中创建工作流

可以使用 vRealize Orchestrator Client 创建和编辑工作流。

步骤

1. 登录到 vRealize Orchestrator 客户端。
2. 选择**库 > 工作流**。
3. 单击**新建工作流**。
4. 输入新工作流的名称，然后单击**创建**。
5. 使用工作流编辑器配置工作流的变量、工作流输入和输出、结构定义结构和展示。
6. 要完成对工作流的编辑，请单击**保存**。

   注　可以在**版本历史记录**选项卡中跟踪对工作流进行的更改。有关详细信息，请参见 vRealize Orchestrator 对象版本历史记录。

后续步骤

可以使用 vRealize Orchestrator 令牌重放功能来优化工作流的性能。有关详细信息，请参见在 vRealize Orchestrator 客户端中使用工作流令牌重放。

编辑父工作流中的工作流和操作

可以直接在 vRealize Orchestrator Client 编辑父工作流中的工作流和操作。

直接从父工作流编辑子工作流和操作有助于简化工作流开发过程。

前提条件

创建一个调用其他工作流和（或）操作的工作流。

步骤

1. 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 工作流**，然后选择工作流。

3 选择**结构定义**选项卡。

4 根据对象类型，在工作流画布中双击**工作流元素**或**操作元素**。

5 编辑对象。

6 要完成对子工作流或操作的编辑，请单击**保存**。

7 要返回到父工作流，请关闭对象编辑器。

vRealize Orchestrator 输入表单设计器

如果工作流需要输入参数，它会打开一个对话框，用户可以在该对话框中输入必需值。可以使用输入表单设计器组织该对话框的内容、布局和展示。

输入表单设计器位于工作流编辑器的**输入表单**选项卡中。该设计器包含导航菜单、设计画布和属性菜单。您可以从左侧菜单将输入和通用元素拖动到设计画布中。在画布中，您可以设置输入参数的位置，将输入参数组织到不同的输入选项卡以及配置输入参数属性。

注 不能在输入表单设计器中使用工作流编辑器的**变量**选项卡中的内容。只能使用**输入/输出**选项卡中的参数。

通用元素

可以将通用元素添加到输入表单设计器中，例如下拉菜单和密码文本框。通用元素不对应于实际输入参数，但可以绑定到输入参数。

在 vRealize Orchestrator 客户端中创建工作流输入参数对话框

可以使用输入表单设计器来创建和自定义工作流输入参数对话框。

前提条件

验证工作流是否拥有输入参数的定义列表。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 工作流**。

3 选择自定义工作流。

4 单击**输入表单**选项卡。

5 （可选） 创建选项卡以在输入对话框中使用。

可以使用选项卡来组织对话框的结构。

6 选择输入参数。

7 编辑输入参数的属性。

有关输入参数属性的详细信息，请参见 vRealize Orchestrator 客户端中的输入参数属性。

8 （可选） 将通用元素添加到画布中，并将其绑定到输入参数。

9 （可选） 添加对输入参数的外部验证。有关详细信息，请参见使用操作验证 vRealize Orchestrator 工作流输入。

10 单击**保存**。

结果

已创建工作流对话框的布局并设置了输入参数的属性。

vRealize Orchestrator 客户端中的输入参数属性

您可以设置参数属性来限制用户在运行 vRealize Orchestrator 工作流时提供的输入参数。

使用 vRealize Orchestrator，可以定义用于量化工作流中所用输入参数值的参数属性。您定义的参数属性会对用户可在 vRealize Orchestrator 工作流中提供的输入参数的类型和值施加限制。

参数属性会验证输入参数并修改文本框在输入参数对话框中的显示方式。有些参数属性可以在参数之间创建依赖关系。

参数属性	描述
标签	设置输入参数标签。
显示类型	设置输入文本框的显示类型。
可见性	设置输入参数的可见性。
只读	将输入文本框设置为只读。
自定义帮助	设置输入参数标志描述。
默认值	设置输入参数的默认值。
步骤	用于数字类型输入。设置每次单击时输入参数值的增加量。
必需	设置输入参数值是否为必填。
正则表达式	使用正则表达式验证输入内容。
最小值	设置参数的最小值或长度。
最大值	设置参数的最大值或长度。
匹配文本框	将输入参数值设置为与其他输入参数的值相匹配。
值源	在**外观**、**值**和**限制**选项卡中设置参数属性的值源。 **注** 可以使用**外部源**导入外部操作的值。筛选可用操作时，可按参数类型进行筛选。

使用操作验证 vRealize Orchestrator 工作流输入

使用外部操作验证自定义工作流的输入。

前提条件

使用输入参数创建自定义工作流。有关详细信息，请参见在 vRealize Orchestrator 客户端中创建工作流。

可以使用输入表单设计器为工作流输入创建外部验证。外部验证使用操作脚本，操作脚本将在输入参数值包含错误时返回一个字符串值。如果输入参数值有效，则外部验证不返回任何内容。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 创建验证操作。

a 导航到**库 > 操作**。

b 单击**新建操作**。

c 在**摘要**选项卡中输入所需信息。

d 输入验证操作输入参数。

> **注** 验证操作输入参数的名称必须与要验证的工作流输入参数的名称相同。

e 在**脚本**选项卡中输入验证操作的脚本。

```
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
```

> **注** 上述脚本是一个简单示例，并且不代表可使用的验证脚本的完整范围。

f 单击**保存**。

3 应用外部验证。

a 导航到**库 > 工作流**。

b 选择自定义工作流。

c 选择**输入表单**选项卡。

d 选择屏幕左上角的剪贴板图标。

e 将 vRealize Orchestrator 验证元素拖动到画布中。

f 选择验证元素，输入验证标签，然后选择验证操作。

g （可选） 创建其他验证元素。

h 单击**保存**。

4 运行工作流。

如果验证遇到错误，则会返回一个字符串。如果验证成功，则不会返回任何内容，并且工作流运行会继续。

结果

已为自定义 vRealize Orchestrator 工作流创建外部验证。

在 vRealize Orchestrator 客户端中请求进行用户交互

工作流在运行过程中可以请求用户进行额外的输入。

如果工作流需要进一步的用户交互，则在用户提供所请求的输入参数之前，工作流会使操作挂起。工作流会定义哪些用户可以提供请求的信息，并相应地发送交互请求。等待用户输入的工作流显示在 vRealize Orchestrator Client 仪表板的**最近的工作流运行**面板中和右上角的通知菜单中。

在 vRealize Orchestrator 客户端中调度工作流

您可以使用调度功能来自动执行 vRealize Orchestrator 工作流运行。

在调度工作流运行时，需要设置已调度任务运行的日期、时间和时间间隔。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 从**库**菜单中选择工作流，然后在工作流面板上，单击**调度**。

3 在**常规**、**调度**和**工作流**类别中配置调度任务参数。

注 **工作流**参数类别仅对需要输入参数的工作流可见。

参数	说明
名称	已调度任务的名称。
描述	关于已调度任务的用途的简短描述。
启动	工作流首次调度运行的日期和时间。
如果是过去的时间，则启动	选择当调度时间为过去的时间时是否启动工作流。选择**是**会立即启动已调度工作流。选择**否**会在下一调度重复周期启动工作流。
调度	设置已调度任务的重复周期模式和事件触发器条目。

参数	说明
结束日期	仅在选择**无重复周期**时可见。设置已调度任务结束的日期和时间。
工作流	输入工作流的输入参数。

4 单击**创建**。

结果

已为工作流创建已调度任务。已调度工作流将显示在**活动 > 已调度**下。您可以单击调度面板上的**删除**来删除已调度任务。

在 vRealize Orchestrator 客户端中编辑已调度任务

可对已调度任务进行编辑，以更改已调度工作流的日期、时间和重复周期等参数。

前提条件

创建已调度工作流任务。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 从**活动 > 已调度**中选择已调度任务。

3 在工作流面板上，单击**编辑**。

4 编辑调度，然后单击**保存**。

注 创建已调度任务时设置的输入参数处于只读状态，无法编辑。要更改这些参数，请为此工作流创建新的已调度任务。

查找工作流中的对象引用

作为工作流开发人员，您可以使用对象引用信息来优化开发生命周期。

在 vRealize Orchestrator Client 中，您可以查找对象引用信息。此功能具有以下两个用途：

- **查找依赖关系**：查找有关工作流中的对象依赖关系的信息。依赖关系可以包括其他工作流、操作、资源元素和配置元素。
- **查找使用情况**：了解所选工作流是否在 vRealize Orchestrator Client 库中的其他工作流中使用。

您可以从工作流编辑器或从 vRealize Orchestrator Client 库的卡视图、列表视图或树视图中访问有关对象引用的信息。有关 vRealize Orchestrator Client 库中各类型内容组织方式的更多信息，请参见 vRealize Orchestrator Client 中的内容组织。

以下过程说明了如何从工作流编辑器访问对象引用。

前提条件

开发至少包含一个对象引用的工作流。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 工作流**，然后选择工作流。

3 要查找有关对象依赖关系的信息，请单击**查找依赖关系**。

> 注 在依赖关系弹出窗口中，您可以从列表中选择引用的对象。选择一个对象后，将打开一个单独的 vRealize Orchestrator Client 选项卡，您可以在其中查看所选对象的详细信息或对其进行编辑。

4 要查找有关使用选定工作流的位置的信息，请单击**查找使用情况**。

管理操作

6

您可以通过添加操作脚本来修改 vRealize Orchestrator 工作流。

vRealize Orchestrator Client 提供预定义操作库和用于自定义操作脚本的操作编辑器。操作表示您在工作流中用作构建块的各个函数。

操作是 JavaScript 函数，操作可使用多个输入参数并拥有单个返回值。操作可以在 vRealize Orchestrator API 中的任何对象上调用，或在您使用插件导入到 vRealize Orchestrator 的任何 API 中的对象上调用。

工作流运行时，操作会从工作流的变量获取输入参数。这些变量可以是工作流的初始输入参数，也可以是工作流中的其他元素在运行时设置的变量。

操作编辑器包含脚本自动补全功能，以及提供可用脚本类型及其文档的 API 资源管理器。

本章讨论了以下主题：

- 在 vRealize Orchestrator 客户端中创建操作
- 运行和调试操作
- Python、Node.js 和 PowerShell 脚本的核心概念
- Python、Node.js 和 PowerShell 脚本的运行时限制

在 vRealize Orchestrator 客户端中创建操作

可以使用 vRealize Orchestrator Client 创建、编辑和删除操作脚本。

在创建操作时，您可以使用以下运行时：

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

> 注 PowerCLI 运行时包括 PowerShell 和以下模块：VMware.PowerCLI、PowerNSX 和 PowervRA。

前提条件

创建 Python、Node.js 或 PowerShell 脚本之前，确认您熟悉用于开发使用这些运行时的 vRealize Orchestrator 兼容脚本的核心概念。请参见 Python、Node.js 和 PowerShell 脚本的核心概念 。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 操作**。

3 单击**新建操作**。

4 在**常规**选项卡中，输入操作的名称和模块名称。

> 注 每个操作的名称和模块名称必须唯一。操作名称必须是有效的 JavaScript 函数。操作名称必须是单个单词，且只能包含字母、数字和美元 ("$") 以及下划线 ("_") 符号。模块名称必须包含用点 (".")字符分隔的单词。

5 （可选） 创建操作的描述、版本号、标记和组权限。

6 在**脚本**选项卡中，添加操作输入，选择输出的返回类型，并编写脚本。

> 注 通过从**类型**下拉菜单中选择 **Zip**，您可以导入外部脚本源和（如适用）依赖关系模块。

7 要完成操作编辑，请单击**保存**。

此时会显示一条消息，指明操作已保存。

后续步骤

要查看用例示例以了解如何使用 vRealize Orchestrator 操作，请参见如何在 vRealize Orchestrator 中使用 Python 集成 Amazon Web Services。

运行和调试操作

您可以通过从操作编辑器中直接运行和调试来改进操作。

可以直接从 vRealize Orchestrator Client 的操作编辑器中运行和调试操作。使用此功能，可以保证操作在集成到工作流时会按预期执行。

在 vRealize Orchestrator 客户端中运行操作

作为工作流设计人员，您需要先运行操作，然后再将它们集成到工作流中。

前提条件

创建操作。请参见在 vRealize Orchestrator 客户端中创建操作。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 操作**，然后选择要运行的操作。

3 单击**运行**。

4 输入必要输入参数，然后单击**运行**。

操作运行完成后，单击**结果/输入**选项卡。如果操作运行遇到错误，将以红色显示在此选项卡上。可以从**操作结果**元素中查看操作运行的详细信息。

注 不会保存操作运行的结果。

vRealize Orchestrator Client 中的调试操作

作为工作流设计人员，您可以通过在脚本中插入断点来调试操作。

vRealize Orchestrator 包含一个内置调试工具，可用来调试脚本和操作的输入属性。在操作编辑器中，可通过在操作的脚本行中插入断点来启动调试过程。

注 内置调试工具仅适用于使用默认 JavaScript 运行时的操作。有关如何调试使用不同运行时的操作脚本的示例，请参见调试 Amazon Web Services 操作。

前提条件

创建操作。请参见在 vRealize Orchestrator 客户端中创建操作。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 操作**，然后选择要调试的操作。

3 在操作编辑器中，将断点添加到要调试的操作脚本的行中。

4 单击**调试**。

5 输入操作的输入参数，然后单击**运行**。

将在调试模式下开始操作运行。

6 当操作运行在到达断点后暂停时，选择以下选项之一：

选项	说明
继续	继续操作运行，直至到达另一个断点或操作运行完成为止。
跳入	跳入当前操作函数。如果调试器无法更深入函数的当前行，则会执行**跳过**操作。
跳过	调试器将继续进入当前函数的下一行。
单步返回	调试器将进入当前函数返回时将执行的行。

7 （可选） 在**调试器**选项卡上，添加表达式。

8 （可选） 在**调试器**选项卡上，编辑变量的值。

Python、Node.js 和 PowerShell 脚本的核心概念

创建要在 vRealize Orchestrator 中使用的脚本时，必须确认脚本的结构和格式正确。

支持的运行时

要开发 vRealize Orchestrator 操作和工作流，您可以使用以下运行时：

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

> 注 PowerCLI 运行时包括 PowerShell 和以下模块：VMware.PowerCLI、PowerNSX 和 PowervRA。

您可以将任何自定义源代码添加到新的运行时，但要接受上下文和输入并从 vRealize Orchestrator 引擎返回结果/将结果返回到该引擎，则必须遵循正确的函数格式。

脚本建议

为了简化脚本任务，可以将**可编辑脚本任务**元素添加到工作流架构中。可以使用 vRealize Orchestrator 操作来执行更复杂的脚本任务。

使用操作可带来以下两个具体优势：

- 可以在工作流中单独创建、更新、导入和导出操作。
- 操作是可以在自己的环境中运行和调试的独立对象，这会使开发过程更为顺畅。请参见运行和调试操作 。

脚本函数要求

脚本函数的默认名称为 **`handler`**。该函数接受两个参数，即上下文和输入。上下文是包含系统信息的映射对象。例如，`vroURL` 可以包含要调用的 vRealize Orchestrator 实例的 URL，而 `executionId` 包含工作流运行的令牌 ID。

输入是包含为操作提供的所有输入的映射对象。例如，如果您在操作中定义了一个名为 `myInput` 的输入，则可以根据您的运行时从输入参数（如 `inputs.myInput` 或 `inputs["myInput"]`）访问该输入。从函数返回的任何内容都是操作的结果。因此，操作的返回类型必须与脚本在 vRealize Orchestrator 中返回的内容类型相对应。如果返回基本类型编号，则操作返回类型必须为数字类型。如果返回字符串，则操作返回类型必须是字符串类型。如果返回复合类型对象，则必须将返回类型映射到 `Properties` 或 `Composite Type`。这些原则同样适用于数组。

对于 Python、Node.js 和 PowerShell 运行时，支持的输入和输出参数类型：

- `String`
- `Number`

- Boolean
- Date
- Properties
- Composite Type

定义条目处理程序

默认情况下，条目处理程序的值为 handler.handler。此值意味着 vRealize Orchestrator 引擎会在 ZIP 软件包中查找名为 handler.py、handler.js 或 handler.ps1 的顶级文件，该文件中包含 handler 函数。函数名称与处理程序文件名之间的任何差异均须反映在条目处理程序的值中。例如，如果主处理程序的名称为 index.js，而函数名为 callMe，则必须将该条目处理程序的值设置为 **index.callMe**。

在外部 IDE 中调试运行时脚本

vRealize Orchestrator 支持在外部 IDE 中调试 Python 和 Node.js 脚本。您无法在外部 IDE 中调试 PowerShell 脚本。

Python、Node.js 和 PowerShell 脚本的运行时限制

某些 Python、Node.js 或 PowerShell 脚本可能会要求您更改 vRealize Orchestrator Client 中的内存和超时值。

vRealize Orchestrator Client 对 Python、Node.js 和 PowerShell 操作脚本使用一组默认的内存和超时值：

- 内存：64 MB
- 超时：180 秒

如果操作脚本超出上述一个或两个默认值，则操作运行将失败。例如，操作脚本可能使用多个第三方依赖关系模块。在这种情况下，默认内存限制 (64 MB) 可能不够。

为了避免因资源不足而导致操作运行失败，请从操作编辑器更改内存和超时值。

注 您还可以考虑将脚本分解为多个可编辑脚本的任务元素，这些元素可以添加到工作流中。

步骤

1 登录到 vRealize Orchestrator Client。

2 导航到**库 > 操作**，然后选择您的操作。

3 选择**脚本**选项卡。

4 在**运行时限制**下，更改内存和超时值。

5 单击**保存**。

6 要测试新的运行时限制，请单击**调试**。

管理配置元素

7

配置元素是一个变量列表，您可以使用这些变量在整个 vRealize Orchestrator 服务器部署中配置各种常量。

您可以使用配置元素使变量可用于 vRealize Orchestrator 服务器上运行的所有工作流、操作和策略。

如果您创建一个软件包，且其中包含使用来自某个配置元素的变量的工作流、操作或策略，则 vRealize Orchestrator 会自动将该配置元素包含到此软件包中。如果将包含某个配置元素的软件包导入到其他 vRealize Orchestrator 服务器中，则还可以导入该配置元素的变量值。例如，如果您创建一个工作流，并且该工作流所需的变量值取决于运行该工作流的 vRealize Orchestrator 服务器，则通过在配置元素中设置这些变量，您可以导出该工作流以供其他 vRealize Orchestrator 服务器使用。因此，配置元素可让您在服务器之间更轻松地交换工作流、操作和策略。

注 无法从导出自 vRealize Orchestrator 5.1 或更低版本的配置元素导入配置元素的变量值。

本章讨论了以下主题：

在 vRealize Orchestrator 客户端中创建配置元素

使用配置元素，您可以在 vRealize Orchestrator 服务器中设置公共变量。在服务器中运行的所有元素都可以使用您在配置元素中设置的变量。

步骤

1. 登录到 vRealize Orchestrator 客户端。
2. 导航到**资产 > 配置**。
3. 选择**新建配置**。
4. 输入配置元素名称。
5. 选择**变量**选项卡。

6 要创建局部变量，请单击**新建**。

a 输入变量名称。

b 选择变量类型。

注 要创建配置变量数组，请选中**数组**复选框。

c （可选） 输入配置变量的值。

d 单击**保存**。

7 要完成配置元素创建，请单击**保存**。

后续步骤

您可以使用配置元素向工作流、操作或策略提供变量。

管理策略

8

策略是用于监视系统活动的事件触发器。策略能够对因特定 vRealize Orchestrator 对象的状态或性能发生变化而引发的预定义事件作出响应。

策略是一组规则、计量器、阈值和事件筛选器，能够在 vRealize Orchestrator 中或在 vRealize Orchestrator 通过插件访问的相关技术中发生特定的预定义事件时，运行特定的工作流和脚本。vRealize Orchestrator 会在策略运行过程中持续评估策略规则。例如，您可以实现策略计量器和阈值用于监视类型为 `VC:HostSystem` 和 `VC:VirtualMachine` 的 vCenter Server 对象的行为。

本章讨论了以下主题：

在 vRealize Orchestrator 客户端中创建并应用策略

可以使用策略监控 vRealize Orchestrator 系统的特定事件活动。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 策略**。

3 选择**新建策略**。

已创建一个空策略。

4 输入策略名称和版本号。

5 选择**变量**选项卡。

6 要创建局部变量，请单击**新建**。

a 输入变量名称。

b 选择变量类型。

注 要创建策略变量数组，请选中**数组**复选框。

c 输入变量值。

注 要导入配置元素变量的值，可以使用**绑定到配置**。

d 单击**保存**。

7 在**定义**选项卡中，添加策略元素并设置事件处理程序。

有关策略元素的详细信息，请参见 vRealize Orchestrator 客户端中的策略元素。

8 单击**保存**。

已配置策略。

后续步骤

要启动策略，请选择该策略并单击**运行**。输入策略运行名称，并在出现提示时输入所需的输入参数。

要查看策略状态，请导航到**活动 > 策略运行**。

vRealize Orchestrator 客户端中的策略元素

可以使用策略元素在发生事件时运行预定义的 vRealize Orchestrator 工作流或脚本。

可以添加策略元素以触发工作流运行或脚本运行来响应对象触发的事件。通过定期事件元素，可以调度工作流运行或脚本运行。通过根元素，可以设置策略的启动或停止行为。策略元素可以具有定义策略元素何时必须运行的事件处理程序。

注 激活策略元素的事件处理程序可以是工作流或操作脚本。如果同时向事件处理程序添加工作流和脚本，则策略会忽略脚本触发器，并且仅使用工作流触发器。

事件处理程序	说明
OnInit	每次启动策略时触发策略元素。
OnExit	每次停止策略时触发策略元素。
OnExecute	由定期事件元素使用。在定期事件元素中指定的时间段内触发策略元素。

注 插入到 vRealize Orchestrator 数据库的技术可以具有唯一的事件处理程序。例如，通过 SNMP 插件，可以在创建基于 SNMP 的策略元素时使用 **OnTrap** 事件处理程序。

可以在策略编辑窗口的**定义**选项卡中配置策略元素。

在 vRealize Orchestrator 客户端中管理策略运行

可以使用 vRealize Orchestrator Client 管理策略优先级和策略的服务器启动行为（例如，何时重新启动 vRealize Orchestrator 服务器）。

前提条件

创建并运行策略。有关详细信息，请参见在 vRealize Orchestrator 客户端中创建并应用策略。

步骤

1. 以管理员身份登录到 vRealize Orchestrator 客户端。
2. 导航到**活动 > 策略运行**。
3. 单击要管理的策略运行。
4. 单击**停止**。

   策略状态将更改为**已停止**。
5. 在**常规**选项卡中，设置策略优先级和服务器启动行为。
6. 要重新启动策略，请单击**运行**。

   策略状态将更改为**正在运行**。

管理资源元素

9

工作流可以使用独立于 vRealize Orchestrator 创建的对象作为属性。要在工作流中将外部对象用作属性，您需要将其作为资源元素导入服务器。

vRealize Orchestrator 工作流可用作资源元素的对象包含图像文件、脚本、XML 模板、HTML 文件等。vRealize Orchestrator 服务器中运行的任意工作流都可以使用您导入 vRealize Orchestrator 的任意资源元素。

将对象作为资源元素导入 vRealize Orchestrator 后，可以在一个集中位置更改对象，并且将这些更改自动传播到使用该资源元素的所有工作流。

资源元素的最大大小为 16 MB。

可以导入、导出、还原、更新和删除资源元素。

管理软件包

10

可使用 vRealize Orchestrator Client 创建、导出和导入软件包。软件包可用于导出工作流对象以在其他 vRealize Orchestrator 实例中使用。

软件包可以包含工作流、操作、策略、配置元素或资源元素。

向软件包中添加元素时，vRealize Orchestrator 会检查依赖关系并将任何从属元素添加到软件包中。例如，如果您添加使用了某些操作或其他工作流的工作流，vRealize Orchestrator 会将这些操作和工作流添加到软件包中。

导入软件包时，服务器会将软件包不同内容元素与匹配的本地元素进行版本比较。比较结果会显示本地元素和导入元素之间的版本差异。用户可以决定是导入软件包，还是选择导入特定元素。

对于在 vRealize Orchestrator Client 中创建的大多数对象，除资源元素之外，软件包是导出和导入这些对象的唯一方法。

软件包通过数字化权限管理来控制接收服务器对软件包内容的使用方式。vRealize Orchestrator 会对软件包签名并进行加密以保护数据。软件包会使用 X509 证书跟踪哪些用户导出并重新分发了元素。

在 vRealize Orchestrator 客户端中创建软件包

您可以导出和导入软件包中的工作流、策略、操作、插件引用、资源元素和配置元素。与软件包对象相关的所有从属元素都会自动添加到软件包中，确保不同版本之间的兼容性。要删除从属元素，必须先移除相关的软件包对象。

对于在 vRealize Orchestrator Client 中创建的大多数对象，除资源元素之外，软件包是导出和导入这些对象的唯一方法。

前提条件

确认 vRealize Orchestrator 服务器包含可添加到软件包中的对象，例如工作流、操作和策略。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**资产 > 软件包**。

3 单击**新建软件包**。

4 在**常规**选项卡中，输入软件包的名称和描述。

注 在 vRealize Orchestrator Client 中命名软件包时，不能使用特殊字符。

5 在**内容**选项卡中，单击**添加**。

6 选择要添加到软件包中的对象，然后单击**添加**。

注 从属元素将自动添加到软件包中，但在软件包创建期间不会在**内容**选项卡中显示。要查看从属元素，请在创建软件包后选择**内容**选项卡。

7 要完成软件包创建，请单击**创建**。

在 vRealize Orchestrator 客户端中导出软件包

可以使用 vRealize Orchestrator Client 将软件包导出到其他 vRealize Orchestrator 环境。

前提条件

创建包含要导出的 vRealize Orchestrator 对象的软件包。有关详细信息，请参见在 vRealize Orchestrator 客户端中创建软件包 。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**资产 > 软件包**。

3 对软件包单击**导出**。

4 （可选） 选择其他导出选项。

选项	说明
将配置属性值添加到软件包	导出配置元素的属性值。
将配置 SecureString 属性值添加到软件包	导出 `SecureString` 配置属性值。
将全局标记添加到软件包	导出全局标记。

5 为导入软件包的用户设置访问权限。

选项	说明
查看内容	用户可以查看软件包内容。
添加到软件包	用户可以将已导入软件包中的内容添加到其他软件包。
编辑内容	用户可以编辑软件包内容。

6 单击**确定**。

注 扩展名为 `.package` 的文件会保存到本地计算机上的默认文件夹中。要设置自定义文件夹，可以在浏览器中更改存储设置。

结果

您即导出了该软件包。现在可以在其他 vRealize Orchestrator 环境中使用导出的对象。

在 vRealize Orchestrator 客户端中导入软件包

可以使用 vRealize Orchestrator Client 导入工作流软件包。通过导入软件包，可以在一台 vRealize Orchestrator 服务器上重用另一台服务器上的对象。

前提条件

- 备份任何已修改的标准 vRealize Orchestrator 对象。
- 在远程服务器上，创建并导出包含您要导入的对象的软件包。

步骤

1. 登录到 vRealize Orchestrator 客户端。
2. 导航到**资产 > 软件包**。
3. 单击**导入**，浏览到要导入的 `.package` 文件，然后单击**打开**。
4. 查看已导入软件包的信息。
   a. **常规**选项卡中包含有关导入的软件包的信息，如名称、描述、包含的项目数和证书信息。

      系统可能会显示提示您需要信任源 vRealize Orchestrator 实例的发布者证书，然后才可导入该文件。

   b. **软件包元素**选项卡中列出导入文件中包含的对象。如果软件包中对象的版本高于服务器上的版本，则系统会选择该版本的对象进行导入。如果要导入低版本的 vRealize Orchestrator 元素，必须手动进行选择。

   c. 如果不想导入软件包中的配置元素的属性值，请取消选中**导入配置属性值**。

   d. 从下拉菜单中选择是否要导入标记。
5. 单击**导入**。

在 vRealize Orchestrator 客户端中进行故障排除

11

可以使用衡量指标、令牌重放、验证和调试来监控 vRealize Orchestrator 实例以及对其进行故障排除。

本章讨论了以下主题：

vRealize Orchestrator 客户端中的衡量指标数据

vRealize Orchestrator 管理员可以使用工作流分析和系统仪表板衡量指标对 vRealize Orchestrator 系统和工作流进行故障排除。

分析功能会收集有关工作流运行的衡量指标数据。默认情况下启用工作流分析。可以在**控制中心 > 扩展属性 > 分析器 8.7.0** 中禁用自动分析。

vRealize Orchestrator Client 中的另一个衡量指标数据源是系统仪表板，可提供系统级别衡量指标。有关详细信息，请参见使用 vRealize Orchestrator 系统仪表板。

在 vRealize Orchestrator 客户端中分析工作流

可以通过分析工作流运行，对 vRealize Orchestrator 环境进行故障排除和优化。

您可以使用 vRealize Orchestrator Client 的分析功能来收集有关工作流运行的有用衡量指标数据。这些数据可用于优化工作流的性能。默认情况下，系统会自动分析工作流运行。可以从 vRealize Orchestrator 控制中心的**扩展属性**页面禁用自动分析，并手动运行分析器。要手动运行分析，请在库中找到工作流并选择**操作 > 分析**。

前提条件

运行工作流。

步骤

1 登录到 vRealize Orchestrator 客户端。

2 导航到**活动 > 工作流运行**。

3 选择工作流运行。

在工作流运行结构定义中，您可以查看有关各个工作流项的数据。数据包括总运行时间、最长运行时间和项目运行数量。可以通过页面右上角的下拉菜单筛选这些信息。

4 选择**性能**选项卡。

此选项卡提供有关工作流运行 CPU 时间、运行时间、令牌大小和工作流项数据的衡量指标数据。

注 如果工作流运行挂起（例如在工作流等待进一步输入时），则 CPU 时间衡量指标仅捕获在完成之前发生的运行时线程。

后续步骤

使用通过分析收集的数据来优化工作流。

使用 vRealize Orchestrator 系统仪表板

作为管理员，您可以使用 vRealize Orchestrator Client 系统仪表板收集有关 vRealize Orchestrator 环境中各节点的有用衡量指标数据。

通过单击 vRealize Orchestrator Client 仪表板页面顶部的**系统**选项卡，可以访问系统仪表板。提供的数据包括：

- 节点状态
- 节点属性
- 群集设置。只能通过系统仪表板查看群集设置。要更改这些设置，请转到 vRealize Orchestrator 控制中心的 **Orchestrator 群集管理**页面。
- 线程信息
- 堆内存
- 非堆内存
- 文件系统使用情况
- 身份验证数据
- Orchestrator 数据库连接池
- 进程输入参数

这些数据可用于监控 vRealize Orchestrator 环境中各个节点的状态以及排除问题。要在各个节点之间导航，请在系统仪表板顶部单击与节点关联的选项卡。

在 vRealize Orchestrator 客户端中使用工作流令牌重放

使用令牌重放功能可以查看工作流运行中的项之间的转换。

令牌重放功能可记录工作流项之间每次转换的上下文信息。对于每个工作流项，令牌重放会记录工作流运行何时开始及何时结束以及工作流项运行结束时哪些变量已更改。对于每个工作流项，令牌重放还引用生成的脚本日志消息。

注 有关工作流项转换的数据存储在 vRealize Orchestrator PostgreSQL 数据库中。删除工作流运行时，这些数据将从数据库中移除。

前提条件

- 在控制中心内启用令牌重放功能。
  a 以 **root** 用户身份登录控制中心。
  b 选择**扩展属性**。
  c 单击 **tokenreplay-8.7.0**。
  d 要启用令牌重放功能，请单击**启用**。
  e 单击**保存**。

  注 vRealize Orchestrator 服务器最多可能需要 5 分钟来刷新扩展。
- 运行工作流。

注 默认情况下，不会为 vRealize Orchestrator 服务器上的所有工作流自动运行令牌重放。可以单独为每个工作流运行令牌重放，也可以在控制中心的**扩展属性**页面中为所有工作流启用令牌重放扩展。

步骤

1 （可选） 为 vRealize Orchestrator 服务器上的所有工作流运行启用令牌重放。

  注 要运行单个令牌重放，而不必在控制中心内启用此功能，请单击工作流编辑器页面上的**通过重放运行**。

  a 以 **root** 用户身份登录控制中心。
  b 选择**扩展属性**。
  c 单击 **tokenreplay-8.7.0**。
  d 要为所有工作流启用令牌重放功能，请确认已启用**对所有工作流运行记录重放**。
  e 单击**保存**。

  注 vRealize Orchestrator 服务器最多可能需要 5 分钟来刷新扩展。

2 以管理员身份登录到 vRealize Orchestrator 客户端。

3 导航到**活动 > 工作流运行**。

4 选择工作流运行。

5 从左侧菜单中选择工作流运行项。

变量选项卡和**日志**选项卡现在显示特定于该工作流项的信息。

验证 vRealize Orchestrator 工作流

vRealize Orchestrator 提供了工作流验证工具。验证工作流有助于识别工作流中的错误，并检查相邻元素之间的数据流动是否正确。

默认情况下，运行工作流时，vRealize Orchestrator 始终会执行工作流验证。

验证工作流时，验证工具会创建一个包含所有错误或警告的列表。在列表中单击某个错误可突出显示包含该错误的工作流元素。

如果在工作流编辑器中运行验证工具，则工具会对其检测到的错误提供快速修复建议。一些快速修复需要其他信息或输入参数，而其他快速修复则会为您解决错误。

工作流验证会检查元素之间的数据绑定和连接。工作流验证不会检查每个元素在工作流中执行的数据处理。因此，如果架构元素中的某个函数不正确，那么即使是有效的工作流也会出现运行错误并产生错误结果。

在 vRealize Orchestrator 客户端中验证工作流和修复验证错误

运行工作流之前必须先对其进行验证。仅当已打开工作流进行编辑时，才能修复验证错误。

前提条件

确认要验证的工作流是否完整，架构元素是否已链接相关绑定是否已定义。

步骤

1 以管理员身份登录到 vRealize Orchestrator 客户端。

2 导航到**库 > 工作流**，然后选择要验证的工作流。

3 单击**编辑**。

4 在顶部菜单中单击**验证**。

如果工作流有效，则会显示确认消息。如果工作流无效，则会显示错误列表。

5 对于无效工作流，单击错误消息并采取适当的步骤以解决问题。

验证工具会通过添加红色图标的方式来突出显示发生错误的架构元素。在可能的情况下，验证工具会显示快速修复操作。

- 如果同意建议的快速修复操作，请单击以执行该操作。
- 如果不同意建议的快速修复操作，请关闭“工作流验证”对话框并手动修复架构元素。

重要事项 始终检查 vRealize Orchestrator 建议的修复是否合适。

例如：建议的操作可能要删除未使用的属性，而事实上该属性却可能是绑定不正确。

6 重复上述步骤，直到消除所有验证错误为止。

结果

您即验证了工作流并修复了验证错误。

后续步骤

您可以运行工作流。

在 vRealize Orchestrator 客户端中调试工作流脚本

可以通过在工作流项的脚本中插入断点来调试工作流运行。

到达断点时，对于如何继续调试操作，您有多种选择。调试工作流结构定义中的元素时，您可以查看有关工作流运行的常规信息，修改工作流变量，添加要监视的表达式以及查看日志消息。

注 在非生产环境中执行所有脚本调试。

步骤

1 以管理员身份登录到 vRealize Orchestrator 客户端。

2 从库中选择工作流。

3 打开工作流结构定义，选择工作流元素，然后单击**脚本**选项卡。

4 要插入断点，请单击行号左边的红色圆圈。

注 只能在使用脚本的工作流元素中插入断点。

5 要在调试模式下运行工作流，请单击**调试**。

如果工作流需要输入参数，则您必须提供。

6 如果工作流运行在到达断点之后暂停，请选择可用选项之一。

选项	描述
继续	继续工作流运行，直至到达另一个断点或工作流运行完成为止。
跳入	可以使用此选项跳入工作流元素。在工作流编辑器中调试工作流时，无法跳入嵌套的工作流元素。
跳过	跳过结构定义中的当前元素并在下一元素上暂停工作流运行。

注 通过单击当前断点，可以指示调试器忽略该断点。这会将断点符号更改为绿色三角形。

7 （可选） 在**调试器**选项卡中，插入要监视的表达式。

可以使用表达式来跟踪特定变量的完成情况。

8 （可选） 在**调试器**选项卡中，修改变量的值。

按结构定义元素调试工作流

作为工作流设计人员，您可以调试各个结构定义元素。

步骤

1 登录到 vRealize Orchestrator Client。

2 导航到**库 > 工作流**，然后选择工作流。

3 选择**结构定义**选项卡。

4 选择要调试的工作流元素，然后单击元素左上角的调试按钮。

注 通过向**工作流元素**结构定义元素添加断点，您可以直接从父工作流调试子工作流。当调试器到达**工作流元素**结构定义元素时，将打开子工作流的结构定义视图。

5 对要调试的任何其他结构定义元素重复上述操作。

6 单击**调试**。

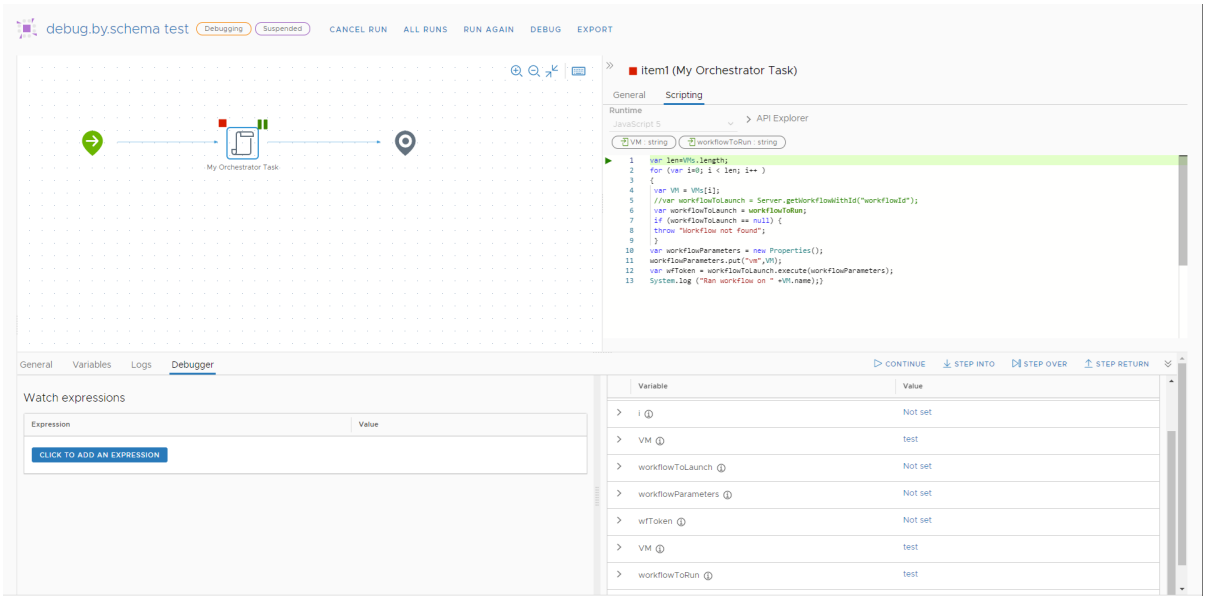
7 输入请求的输入参数值，然后单击**运行**。

工作流运行开始，并在调试器到达具有断点的结构定义元素时暂停。

8 在断点处时，选择以下选项之一：

选项	描述
继续	继续工作流运行，直至到达另一个断点或工作流运行完成为止。
跳入	跳入当前工作流函数。如果调试器无法更深入函数的当前行，则会执行**跳过**操作。
跳过	调试器将继续进入当前函数的下一行。
单步返回	调试器将进入当前函数返回时将执行的行。

9 （可选） 在**变量**选项卡上，编辑工作流变量的值。



为 Python 包配置 Photon OS 容器

根据用于编译 Python 脚本的操作系统 (Operating System, OS)，在将相关的 ZIP 存档导入到 vRealize Orchestrator Client 后，工作流或操作可能会失败。

vRealize Orchestrator 中用于 Python 的运行时容器的操作系统基于 Photon 3.0。为其他操作系统（例如 Linux）编译的 Python 脚本包与运行时容器不兼容。当您尝试在 vRealize Orchestrator 工作流或操作中使用 Python 脚本时，此问题可能会导致 Python 脚本运行失败。在此类场景中，您将在日志中收到以下错误消息：

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

要解决此问题，必须将所需的 Python 软件包安装到 Photon OS 容器文件夹中。

前提条件

安装 Docker。请参见获取 Docker。

步骤

1 导航到 Python 脚本的父文件夹。

2 通过将容器文件夹挂载到父文件夹，创建含有基础 Photon 映像的容器。

注 以下脚本是一个单独的 Docker 命令，您必须完整运行该命令，才能创建适当的容器。

```
docker run -ti -v
$(pwd)/<name_of_folder_that_containes_your_python_script>/:/
<name_of_folder_that_containes_your_python_script>
photon:3.0
```

3 在容器中安装 Python。

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

4 导航到包含您的 Python 脚本的容器文件夹。

5 添加您的 Python 脚本和软件包。

注 将 Python 脚本所需的软件包安装到 `lib` 文件夹中。

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

6 退出容器，然后导航到已挂载到容器的本地文件夹。

7 将所有相关文件和文件夹压缩为一个 ZIP 存档。

8 将该 ZIP 存档导入到 vRealize Orchestrator Client，并通过在操作中运行该脚本来对其进行验证。