

VMware vRealize Orchestrator 클라이언트 사 용

2020년 10월 6일

vRealize Orchestrator 8.2

다음 VMware 웹 사이트에서 최신 기술 문서를 확인할 수 있습니다.

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

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

VMware 코리아
서울시 강남구
영동대로 517
아셈타워 13층
(우) 06164
전화: +82 2 3016 6500
팩스: +82 2 3016 6501
www.vmware.com/kr

목차

1	VMware vRealize Orchestrator 클라이언트 사용	5
2	VMware vRealize Orchestrator 클라이언트	6
	vRealize Orchestrator 클라이언트 사용량 대시보드	7
	vRealize Orchestrator Client의 콘텐츠 구성	7
	vRealize Orchestrator 클라이언트에서 폴더 또는 하위 폴더 생성	9
3	vRealize Orchestrator Client 설정	12
	vRealize Orchestrator 클라이언트 역할 및 그룹 관리	12
	vRealize Orchestrator Client에서 역할 할당	13
	vRealize Orchestrator Client에서 그룹 생성	14
	vRealize Automation에서 vRealize Orchestrator 클라이언트 역할 구성	15
	vRealize Orchestrator 개체 버전 기록	16
	워크플로를 이전 버전으로 복원	16
	워크플로 버전 간의 시각적 비교	17
	Git을 사용하여 vRealize Orchestrator 콘텐츠 인벤토리를 이전 상태로 재설정	18
4	vRealize Orchestrator 사용 사례	19
	Python을 사용하여 vRealize Orchestrator에서 Amazon Web Services를 통합하는 방법	19
	초기 Python 스크립트 생성	20
	Amazon Web Services 작업 생성	21
	Amazon Web Services 작업 디버깅	22
	Amazon Web Services 작업 업데이트	25
	Git 분기를 사용하여 내 vRealize Orchestrator 개체 인벤토리를 관리하는 방법	26
	GitLab 환경 준비	27
	Git 저장소에 대한 연결 구성	27
	변경 내용을 Git 저장소로 푸시	29
	타사 모듈을 사용하여 vRealize Automation 프로젝트 API를 호출하는 방법	31
	vRealize Automation 프로젝트 API를 호출하는 Python 스크립트 생성	31
	vRealize Automation 프로젝트 API를 호출하는 Node.js 스크립트 생성	33
	vRealize Automation 프로젝트 API를 호출하는 PowerShell 스크립트 생성	35
5	워크플로 관리	39
	vRealize Orchestrator 워크플로 라이브러리의 표준 워크플로	40
	vRealize Orchestrator 클라이언트에서 워크플로 생성	40
	상위 워크플로에서 워크플로 및 작업 편집	40

vRealize Orchestrator 입력 양식 디자이너	41
vRealize Orchestrator 클라이언트에서 워크플로 입력 매개 변수 대화 상자 생성	41
vRealize Orchestrator 클라이언트의 입력 매개 변수 속성	42
작업을 사용하여 vRealize Orchestrator 워크플로 입력 검증	43
vRealize Orchestrator 클라이언트에서 사용자 상호 작용 요청	44
vRealize Orchestrator 클라이언트에서 워크플로 예약	45
vRealize Orchestrator 클라이언트에서 예약된 작업 편집	45
6 작업 관리	47
vRealize Orchestrator 클라이언트에서 작업 생성	47
작업 실행 및 디버깅	48
vRealize Orchestrator 클라이언트에서 작업 실행	49
vRealize Orchestrator 클라이언트에서 작업 디버깅	49
Python, Node.js 및 PowerShell 스크립트에 대한 핵심 개념	50
Python, Node.js 및 PowerShell 스크립트에 대한 런타임 제한	51
7 구성 요소 관리	53
vRealize Orchestrator 클라이언트에서 구성 요소 생성	53
8 정책 관리	55
vRealize Orchestrator 클라이언트에서 정책 생성 및 적용	55
vRealize Orchestrator 클라이언트의 정책 요소	56
vRealize Orchestrator 클라이언트에서 정책 실행 관리	57
9 리소스 요소 관리	58
10 패키지 관리	59
vRealize Orchestrator 클라이언트에서 패키지 생성	59
vRealize Orchestrator 클라이언트에서 패키지 내보내기	60
vRealize Orchestrator 클라이언트에서 패키지 가져오기	61
11 vRealize Orchestrator 클라이언트에서 문제 해결	63
vRealize Orchestrator 클라이언트의 메트릭 데이터	63
vRealize Orchestrator 클라이언트에서 워크플로 프로파일링	63
vRealize Orchestrator 시스템 대시보드 사용	64
vRealize Orchestrator Client에서 워크플로 토큰 재생 사용	65
vRealize Orchestrator 워크플로 유효성 검사	66
vRealize Orchestrator 클라이언트에서 워크플로 검증 및 검증 오류 수정	66
vRealize Orchestrator 클라이언트에서 워크플로 스크립트 디버그	67
스키마 요소별 워크플로 디버깅	68

VMware vRealize Orchestrator 클라이언트 사용

1

"VMware vRealize Orchestrator 클라이언트 사용"에서는 워크플로 자동화 기능 및 vRealize Orchestrator Client의 기능에 대한 정보를 제공합니다.

대상 사용자

이 정보는 vRealize Orchestrator 워크플로를 실행하고 관리하는 데 도움이 되는 도구를 찾고 있는 숙련된 시스템 관리자를 위해 작성되었습니다.

VMware vRealize Orchestrator 클라이언트

2

vRealize Orchestrator Client를 사용하여 vRealize Orchestrator 서비스 및 개체를 관리합니다.

vRealize Orchestrator Client를 사용하여 vRealize Orchestrator 개체를 생성하고 관리할 수 있습니다. vRealize Orchestrator Client는 https://your_orchestrator_FQDN/orchestration-ui에서 찾을 수 있습니다.

REST API 통신

vRealize Orchestrator Client는 vRealize Orchestrator 제어 센터 서버에서 실행됩니다. 클라이언트는 REST 프록시를 통해 vRealize Orchestrator REST API와 통신합니다.

워크플로우 관리

vRealize Orchestrator Client에서 워크플로를 생성, 편집, 스케줄링, 실행 및 삭제합니다.

작업 관리

vRealize Orchestrator Client에서 작업을 생성, 편집 및 삭제합니다. 작업 편집기는 vRealize Orchestrator API 탐색기에 포함된 일반 스크립트 요소에 대한 자동 완성을 지원합니다.

정책 관리

vRealize Orchestrator Client에서 정책을 생성, 편집, 실행 및 삭제합니다.

구성 관리

vRealize Orchestrator Client에서 구성 요소를 생성, 실행 및 삭제합니다.

리소스 관리

vRealize Orchestrator Client에서 리소스 요소를 내보내고, 가져오고, 업데이트합니다.

Git 통합

Git 저장소에 대한 통합을 생성하고 통합을 사용하여 여러 배포에서 워크플로 및 기타 vRealize Orchestrator의 개발을 관리합니다. [Git 분기를 사용하여 내 vRealize Orchestrator 개체 인벤토리를 관리하는 방법](#)의 내용을 참조하십시오.

메트릭 데이터

vRealize Orchestrator Client 시스템 대시보드 및 프로파일링 기능을 사용하여 vRealize Orchestrator 환경 및 워크플로에 대한 유용한 메트릭 데이터를 수집합니다.

패키지 관리

vRealize Orchestrator Client를 통해 vRealize Orchestrator 개체가 포함된 패키지를 생성하고, 삭제하고, 내보내고, 가져옵니다.

사용 권한 관리

관리자 권한이 있는 사용자는 vRealize Orchestrator Client의 사용자에게 역할을 할당하고 사용자 그룹에 추가할 수 있습니다.

API 탐색기

vRealize Orchestrator Client에서 사용 가능한 API 명령을 살펴봅니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트 사용량 대시보드](#)
- [vRealize Orchestrator Client의 콘텐츠 구성](#)

vRealize Orchestrator 클라이언트 사용량 대시보드

vRealize Orchestrator Client 대시보드는 vRealize Orchestrator Client 워크플로를 모니터링 및 관리하고 문제를 해결하는 데 유용한 도구를 제공합니다.

vRealize Orchestrator Client 대시보드의 정보는 5개 패널에 분산되어 있습니다.

창	설명
워크플로 실행	실행 중인 워크플로, 실행 대기 중인 워크플로 및 실패한 워크플로로 실행의 수에 대한 시각적 데이터를 제공합니다.
즐거찾기 워크플로	즐거찾기에 추가된 워크플로를 표시합니다.
입력 대기 중	추가 사용자 상호 작용이 필요한 보류 중인 워크플로 실행을 표시합니다. 이러한 워크플로는 UI의 오른쪽 상단 모서리에 있는 알림 메뉴에도 표시됩니다.
최근 워크플로 실행	최근 워크플로 실행을 관리합니다. 워크플로의 실행의 이름, 상태, 시작 날짜 및 종료 날짜를 표시합니다.
주의 필요	실패한 워크플로 실행 및 워크플로 실행 메트릭을 표시합니다.

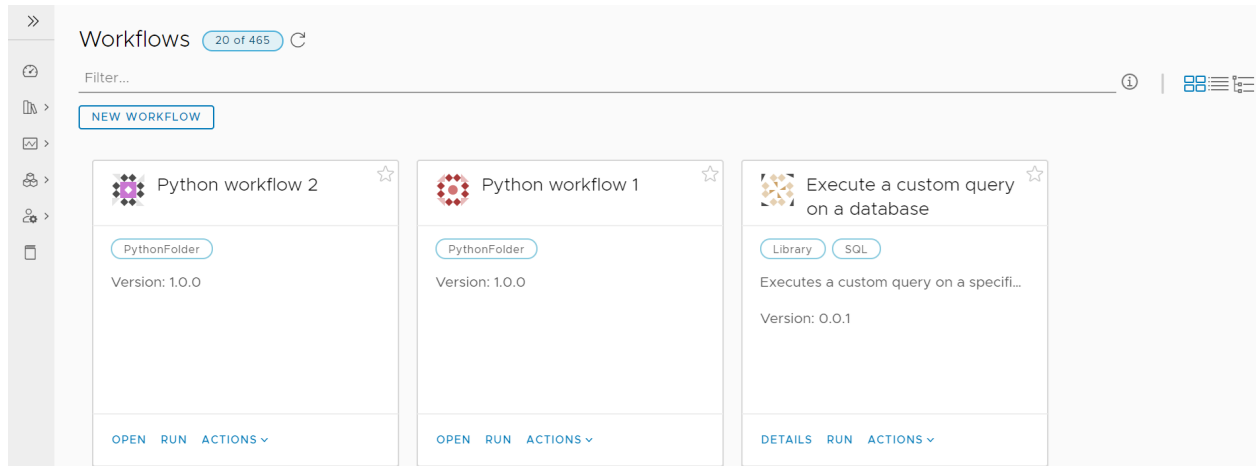
vRealize Orchestrator Client의 콘텐츠 구성

vRealize Orchestrator Client에 vRealize Orchestrator 개체 인벤토리가 표시되는 방식을 관리합니다.

vRealize Orchestrator Client는 워크플로, 작업, 정책, 리소스, 구성과 같은 개체에 대해 카드 보기, 목록 보기 및 트리 보기의 세 가지 다른 보기 유형을 지원합니다. 페이지의 오른쪽 상단에서 현재 보기 유형을 변경할 수 있습니다.

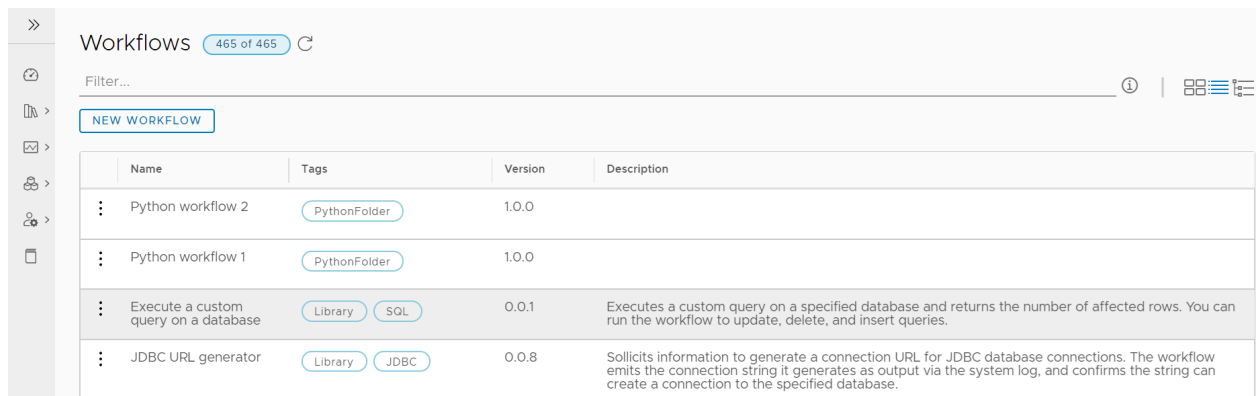
카드 보기

카드 보기는 vRealize Orchestrator Client에 사용되는 기본 보기 유형입니다. 워크플로와 같은 개별 인벤토리 개체에 대한 정보는 별도의 카드 요소에 표시됩니다.



목록 보기

목록 보기는 목록으로 구성된 vRealize Orchestrator 개체에 대한 정보를 표시합니다. 개체에 수행할 수 있는 작업에 대한 자세한 내용을 보려면 개체 왼쪽에 있는 세로 줄임표 아이콘을 클릭합니다.



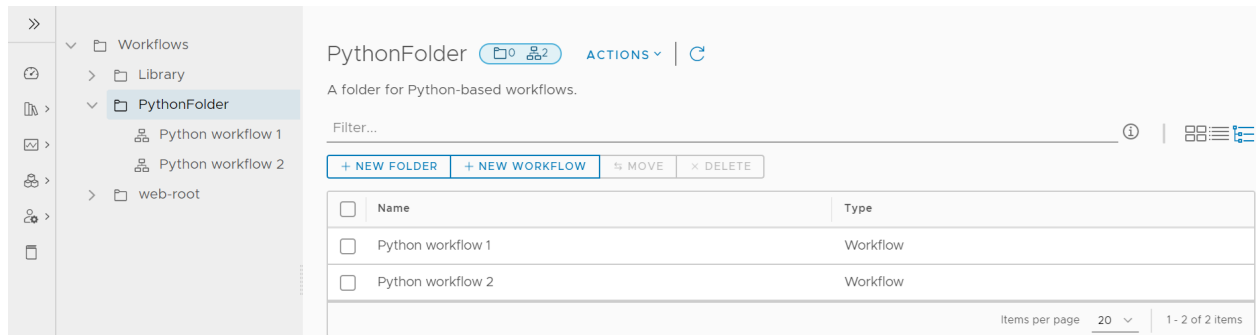
트리 보기

트리 보기의 계층형 폴더 아래에서 개체 인벤토리를 구성할 수 있습니다. 각 vRealize Orchestrator 개체 유형에는 루트 수준 폴더가 있습니다. 루트 폴더 아래에는 워크플로와 같은 새 개체를 생성할 수 없습니다. 루트 폴더 아래에 구성된 별도의 폴더를 생성해야 합니다. 각 폴더에는 콘텐츠 필터와 같은 콘텐츠를 관리하는 데 도움이 되는 도구가 포함되어 있습니다.

참고 각 폴더에는 별도의 콘텐츠 필터가 있습니다. 여러 폴더에 걸쳐 콘텐츠를 필터링 할 수 없습니다.

폴더에 대한 자세한 내용은 [vRealize Orchestrator 클라이언트에서 폴더 또는 하위 폴더 생성 항목을 참조하십시오.](#)

참고 트리 보기에서 개체를 선택하면 읽기 전용 모드로 열립니다. 워크플로 변수 또는 워크플로 스키마와 같은 개체 콘텐츠를 편집하려면 상위 옵션 메뉴에서 **편집**을 클릭합니다.




vRealize Orchestrator 클라이언트에서 폴더 또는 하위 폴더 생성

계층적 폴더 구조를 사용하여 vRealize Orchestrator 개체를 구성합니다.

폴더 및 하위 폴더를 생성하여 다음과 같은 유형의 vRealize Orchestrator 개체를 구성할 수 있습니다.

- 워크플로
- 작업
- 정책
- 구성 요소
- 리소스 요소

절차


- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 왼쪽 탐색 창에서 **워크플로**와 같은 개체 페이지를 선택합니다.
- 3 오른쪽 상단에서 트리 보기 아이콘()을 선택합니다.
- 4 (선택 사항) 하위 폴더를 생성하려면 왼쪽의 트리 보기에서 상위 폴더를 선택합니다.
- 5 **새 폴더**를 클릭합니다.
- 6 이름과 설명을 입력하고 **저장**을 클릭합니다.
- 7 새로 생성한 폴더에 개체 또는 하위 폴더를 추가합니다.
- 8 (선택 사항) 폴더 이름을 편집하려면 **작업 > 편집**을 선택합니다.

vRealize Orchestrator Client에서 개체 및 폴더 이동

컨텐츠를 다른 폴더로 옮겨 vRealize Orchestrator 컨텐츠를 재구성합니다.

작업 모듈 간에 작업을 이동하거나 개체를 루트 폴더로 이동할 수 없습니다. 루트 폴더에는 기본 개체 폴더와 하위 폴더가 포함되지만 개체를 저장하는 데에는 사용할 수 없습니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 왼쪽 탐색 창에서 **워크플로**와 같은 개체 페이지를 선택합니다.
- 3 오른쪽 상단에서 트리 보기 아이콘()을 선택합니다.
- 4 트리 보기를 확장하고 이동할 개체 또는 폴더를 선택합니다.
- 5 개체 또는 폴더를 새 상위 폴더로 끌어옵니다.


참고 개체 편집기에서 개체를 새 폴더로 직접 이동할 수도 있습니다. **요약** 탭에서 **폴더 선택**을 클릭하고 개체의 새 상위 폴더를 선택합니다. 또 다른 이동 옵션은 폴더 페이지의 테이블에서 개체를 선택하는 것입니다. 이 옵션은 여러 vRealize Orchestrator 개체를 포함하는 배치 이동 작업을 수행하는 데 유용합니다.

vRealize Orchestrator Client에서 폴더 또는 하위 폴더 삭제

사용되지 않는 폴더 또는 하위 폴더를 vRealize Orchestrator Client에서 삭제합니다.

각 vRealize Orchestrator 개체 유형의 해당 루트 수준 폴더는 삭제할 수 없습니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 왼쪽 탐색 창에서 **워크플로**와 같은 개체 페이지를 선택합니다.
- 3 오른쪽 상단에서 트리 보기 아이콘()을 선택합니다.
- 4 삭제하려는 폴더 옆의 확인란을 선택합니다.

참고 하위 폴더를 삭제하려면 트리 보기에서 상위 폴더를 선택한 후 확인란을 선택합니다.

- 5 **삭제**를 클릭합니다.
- 6 선택한 폴더가 비어 있으면,
 - a 폴더를 삭제할지 확인합니다.
 - b **삭제**를 클릭합니다.
- 7 선택한 폴더에 vRealize Orchestrator Client 개체 또는 하위 폴더가 포함되어 있으면,
 - a 폴더를 삭제할지 확인합니다.
 - b **삭제**를 클릭합니다.

'your_folder_name' 항목을 삭제할 수 없습니다. 폴더 'your_folder_name'이(가) 비어 있지 않습니다.라는 메시지가 표시됩니다.

- c 폴더와 폴더의 콘텐츠를 모두 삭제하려면 **강제 삭제**를 클릭합니다.
- d 폴더를 삭제할지 확인하고 **삭제**를 클릭합니다.

참고 폴더 메뉴에 포함된 테이블에서 여러 개체를 선택하여 일괄 삭제를 수행할 수도 있습니다.

vRealize Orchestrator Client 설정

3

vRealize Orchestrator Client의 기능을 최대한 활용하려면 사용자 권한을 구성하고 개체 관리를 위해 버전 기록을 사용하는 방법을 알아야 합니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트 역할 및 그룹 관리](#)
- [vRealize Orchestrator 개체 버전 기록](#)

vRealize Orchestrator 클라이언트 역할 및 그룹 관리

관리자는 vRealize Orchestrator Client를 사용하여 vRealize Orchestrator 기능 및 콘텐츠에 대한 사용자 역할 및 그룹 사용 권한을 설정할 수 있습니다.

vRealize Orchestrator 인스턴스가 인증되면 관리자는 기능 및 콘텐츠에 대한 액세스를 제어하는 사용 권한을 설정할 수 있습니다. vRealize Orchestrator Client의 사용 권한은 역할 관리와 그룹 사용 권한으로 구분됩니다. 역할 관리를 사용하여, 사용자가 보고 사용할 수 있는 vRealize Orchestrator Client 기능을 제어합니다. 그룹 사용 권한을 사용하여, 사용자가 보고 사용할 수 있는 vRealize Orchestrator Client 콘텐츠를 제어합니다. 그룹 사용 권한에서 다루는 콘텐츠 액세스에는 워크플로, 작업, 정책, 구성 요소 및 리소스 요소가 포함됩니다. 그룹을 사용하여 사용자를 공통 프로젝트로 구성할 수 있습니다. 예를 들어 사용자 지정 vRealize Orchestrator 플러그인을 개발하는 사용자가 포함된 그룹을 생성할 수 있습니다.

참고 표준 워크플로 및 작업과 같은 미리 구성된 vRealize Orchestrator 콘텐츠에 대한 액세스는 그룹 사용 권한을 통해 달리 구성되지 않는 한 모든 사용자 사이에 공유됩니다.

역할

클라이언트 측 역할 관리는 vRealize Automation 라이선스를 사용하며 vSphere에서 인증된 vRealize Orchestrator 인스턴스에서만 사용할 수 있습니다. vRealize Automation 인증을 사용하는 배포의 경우, vRealize Automation의 ID 및 액세스 관리 기능을 사용해야 합니다. [vRealize Automation](#)에서 [vRealize Orchestrator 클라이언트 역할 구성](#)의 내용을 참조하십시오.

역할	설명
관리자	특정 그룹에서 생성된 콘텐츠를 포함하여 모든 vRealize Orchestrator Client 기능 및 콘텐츠에 액세스 할 수 있습니다. 사용자 역할 설정, 그룹 생성 및 삭제, 그룹에 사용자 추가를 담당합니다. 참고 vRealize Orchestrator 인증에 사용되는 vRealize Automation 환경의 테넌트 관리자는 기본적으로 관리자 권한을 갖습니다.
워크플로 디자이너	자신의 vRealize Orchestrator Client 콘텐츠를 생성, 실행, 편집 및 삭제할 수 있습니다. 자신의 콘텐츠를 할당된 그룹에 추가할 수 있습니다. vRealize Orchestrator Client의 관리 및 문제 해결 기능에 액세스할 수 없습니다.

참고 미리 정의된 역할이 없는 vRealize Automation 사용자는 vRealize Orchestrator Client에 여전히 로그인할 수 있지만 클라이언트 기능에 대한 액세스가 제한됩니다. 그룹에 속해 있는 사용자는 해당 그룹과 관련된 콘텐츠를 보고 실행할 수 있습니다.

그룹

vRealize Orchestrator Client의 그룹 사용 권한을 사용하여 사용자 지정 플러그인 개발과 같은 공통 vRealize Orchestrator에서 작업하는 여러 사용자를 연결할 수 있습니다.

그룹 사용자 사용 권한	설명
실행 및 편집	vRealize Automation 라이선스를 사용하는 vRealize Orchestrator 인스턴스에만 사용할 수 있습니다. 그룹에서 사용할 vRealize Orchestrator 개체를 생성, 편집, 추가 및 실행할 수 있습니다.
실행	그룹에 포함된 vRealize Orchestrator 개체를 보고 실행할 수 있습니다.

참고 그룹 사용 권한은 vRealize Orchestrator Client의 역할 관리 시스템에 연결됩니다. 예를 들어 미리 정의된 역할이 없는 사용자에게 **실행 및 편집** 권한이 있을 수 있지만, 콘텐츠를 생성, 편집 및 추가할 수 없으면 자신의 콘텐츠 또는 그룹 콘텐츠만 보고 실행할 수 있습니다.

vRealize Orchestrator Client에서 역할 할당

관리자는 vRealize Orchestrator Client에 사용자를 추가하고 해당 사용자가 보고 사용할 수 있는 기능을 설정할 수 있습니다.

역할 관리는 vRealize Orchestrator ID 제공자에서 vRealize Orchestrator Client의 기능에 대한 사용자 액세스를 제어합니다. 역할 관리에는 vRealize Orchestrator Client 사용자 인터페이스와 API 기능이 둘 다 포함됩니다.

참고 클라이언트 측 역할 관리는 vRealize Automation 라이선스를 사용하며 vSphere에서 인증된 vRealize Orchestrator 인스턴스에서만 사용할 수 있습니다. vRealize Automation으로 인증된 vRealize Orchestrator에 역할을 할당하는 방법은 [vRealize Automation에서 vRealize Orchestrator 클라이언트 역할 구성](#) 항목을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 2 **관리 > 역할 관리**로 이동합니다.
- 3 **추가**를 클릭합니다.
- 4 vRealize Orchestrator Client에 추가하려는 사용자 또는 그룹을 검색합니다.
- 5 사용자의 역할을 선택합니다. 자세한 내용은 [vRealize Orchestrator 클라이언트 역할 및 그룹 관리](#)의 내용을 참조하십시오.
- 6 **저장**을 클릭합니다.

vRealize Orchestrator Client에서 그룹 생성

관리자는 그룹을 사용하여 사용자가 vRealize Orchestrator Client에서 보고 액세스할 수 있는 vRealize Orchestrator 콘텐츠를 설정할 수 있습니다.

vRealize Orchestrator Client를 사용하여 vRealize Orchestrator 워크플로, 작업, 정책, 구성 요소, 리소스 요소 및 패키지에 대한 그룹 사용 권한을 설정할 수 있습니다.

참고 vSphere로 인증된 vRealize Orchestrator 인스턴스의 사용자는 **실행** 그룹 사용 권한만 가질 수 있습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 2 **관리 > 그룹**으로 이동합니다.
- 3 **새 그룹**을 클릭합니다.
- 4 **요약** 탭에서 그룹에 대한 설명과 이름을 추가합니다.
- 5 **사용자** 탭에서 **추가**를 클릭합니다.
 - a 그룹에 추가할 사용자를 검색합니다.
 - b 사용자에게 그룹 사용 권한을 할당합니다.
 - c **추가**를 클릭합니다.

6 항목 탭에서 vRealize Orchestrator 개체를 그룹에 추가합니다.

참고 개체가 vRealize Orchestrator Client에서 생성될 때 기존 그룹에 개체를 추가할 수도 있습니다. 개체를 추가하려면 개체 편집기의 **요약/일반** 탭에 있는 **액세스할 수 있는 기준** 드롭다운 메뉴에서 그룹을 선택합니다.

7 저장을 클릭합니다.

vRealize Automation에서 vRealize Orchestrator 클라이언트 역할 구성

vRealize Automation의 **ID 및 액세스 관리** 페이지에서 vRealize Orchestrator Client에 대한 서비스 역할을 할당할 수 있습니다. 서비스 역할은 vRealize Automation으로 인증된 내장형 vRealize Orchestrator Client 및 독립형 vRealize Orchestrator 인스턴스에 대해 할당할 수 있습니다.

vRealize Orchestrator 서비스 역할은 내장된 vRealize Orchestrator Client 사용자가 액세스할 수 있는 기능을 관리합니다. vRealize Orchestrator 역할에 대한 자세한 내용은 [vRealize Orchestrator 클라이언트 역할 및 그룹 관리](#) 항목을 참조하십시오.

참고 vRealize Automation 라이선스를 사용하며 vSphere로 인증된 독립형 vRealize Orchestrator 인스턴스는 vRealize Orchestrator Client에서 직접 역할을 할당할 수 있습니다. [vRealize Orchestrator Client에서 역할 할당](#)의 내용을 참조하십시오.

사전 요구 사항

- 유효한 vIDM 인스턴스에서 적절한 사용자 및 그룹을 가져왔는지 확인합니다.
- 사용자에게 vRealize Orchestrator 서비스 역할을 할당하기 전에 사용자에게 vRealize Automation에서 할당된 조직 역할이 있는지 확인합니다. "vRealize Automation 관리"에서 "vRealize Automation에서 사용자 및 그룹 관리"를 참조하십시오.

절차

- 1 오른쪽 상단 머리글 드롭다운 메뉴에서 **ID 및 액세스 관리** 옵션을 선택합니다.
- 2 **활성 사용자** 탭에서 vRealize Orchestrator에 할당할 사용자의 이메일 주소를 검색합니다.
- 3 사용자 옆의 확인란을 선택하고 **역할 편집**을 클릭합니다.
- 4 **서비스 액세스 추가**를 클릭합니다.
- 5 왼쪽 드롭다운 메뉴에서 **Orchestrator**를 선택합니다.
- 6 오른쪽 드롭다운 메뉴에서 사용자에게 할당할 역할을 선택합니다.
- 7 **저장**을 클릭합니다.

vRealize Orchestrator 개체 버전 기록

vRealize Orchestrator Client는 각 vRealize Orchestrator 개체에 대한 버전 기록 레코드를 유지합니다. 버전 기록을 사용하면 서로 다른 vRealize Orchestrator 개체 버전을 비교하고 이전 버전으로 되돌릴 수 있습니다.

각 vRealize Orchestrator 개체를 저장할 때 vRealize Orchestrator에서 개체에 대한 버전 기록 레코드를 생성합니다. 이후에 vRealize Orchestrator 개체를 변경하면 새 버전 기록 레코드가 생성됩니다. 이전 버전의 기록 레코드는 보존되며 개체에 대한 변경 내용을 추적하고 개체를 이전 버전으로 되돌리는 데 사용할 수 있습니다. 개체를 이전 버전으로 되돌리면 새 버전 기록 레코드가 생성됩니다.

vRealize Orchestrator Client는 다음 vRealize Orchestrator 개체의 버전 기록을 추적합니다.

- 워크플로
- 작업
- 패키지
- 정책
- 구성 요소

개체의 버전 기록은 개체 편집기 페이지의 **버전 기록** 탭에서 액세스할 수 있습니다. 개체를 다른 사용자와 동시에 편집하려고 하면 병합 충돌이 발생할 수 있습니다. 병합 충돌을 해결하려면 오류 메시지 오른쪽에 있는 **해결**을 클릭합니다. **충돌 해결** 창에서 다음 세 가지 옵션을 사용할 수 있습니다.

- **다른 사용자의 변경 내용 사용.** 다른 사용자의 변경 내용을 사용하여 병합 충돌을 해결합니다.
- **내 변경 내용 사용.** 사용자 자신의 변경 내용을 사용하여 병합 충돌을 해결합니다.
- **해결.** 표시된 변경 모델을 편집하여 병합 충돌을 해결합니다. 제공된 모델이 잘못된 경우 이 옵션은 사용할 수 없습니다.

워크플로를 이전 버전으로 복원

워크플로를 이전에 저장된 버전으로 복원할 수 있습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리 > 워크플로**로 이동하여 워크플로를 선택합니다.
- 3 **버전 기록** 탭을 선택합니다.
- 4 버전 간의 비교를 보려면 워크플로 버전을 하나 선택하고 **비교 대상** 드롭다운 메뉴에서 다른 버전을 선택합니다.

현재 워크플로 버전과 선택한 워크플로 버전 간의 차이점이 표시된 창이 나타납니다.

- 5 워크플로를 다른 버전으로 복원하려면 **복원**을 클릭합니다.

워크플로 상태가 선택한 버전의 상태로 되돌려집니다.

참고 그래픽 차이 도구 보기에서 워크플로 버전을 복원할 수도 있습니다. [워크플로 버전 간의 시각적 비교](#)의 내용을 참조하십시오.

워크플로 버전 간의 시각적 비교

그래픽 차이 도구를 사용하여 워크플로 버전 간의 변경 내용을 비교합니다.

기본적으로 vRealize Orchestrator 버전 기록은 YAML 형식으로 워크플로 버전 간 차이를 표시합니다. 서로 다른 워크플로 버전 간에 시각적 비교를 수행할 수도 있습니다. 다음에 대한 변경 내용을 볼 수 있습니다.

- 버전 번호, 워크플로 설명과 같은 일반 워크플로 정보.
- 워크플로에 사용된 변수.
- 워크플로의 입력 및 출력 매개 변수.
- 워크플로 스키마.

사전 요구 사항

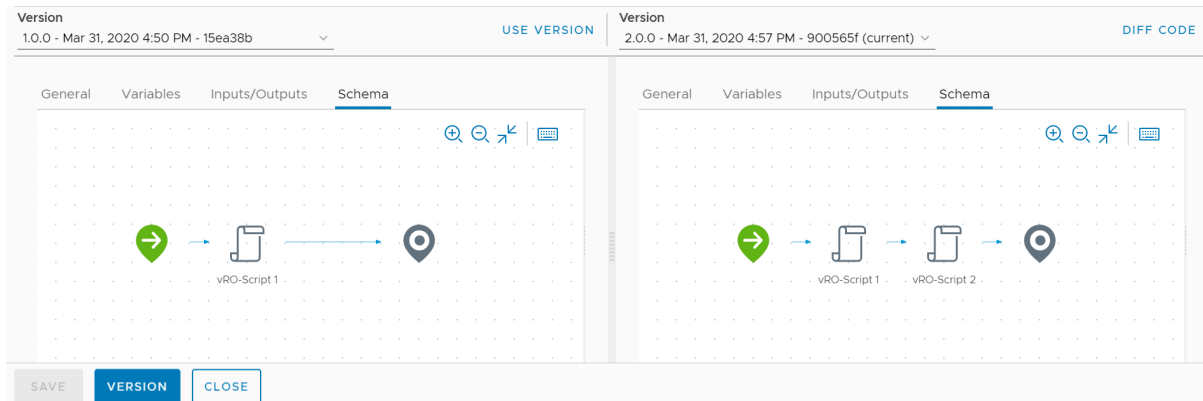
워크플로를 만듭니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 **라이브러리 > 워크플로**로 이동한 후 워크플로 중 하나를 선택합니다.
- 3 워크플로의 콘텐츠를 편집합니다.
예를 들어 **스키마** 탭에서 다른 **스크립팅 가능한 작업** 요소를 추가할 수 있습니다.
- 4 **저장**을 클릭합니다.
- 5 **버전 기록** 탭을 선택합니다.

6 오른쪽 상단에서 **시각적 차이점**을 선택합니다.

이제 선택한 두 워크플로 버전 간에 시각적 비교를 수행할 수 있습니다. **버전** 드롭다운 메뉴에서 비교할 버전을 선택할 수 있습니다.



7 (선택 사항) **버전 사용**을 선택하여 워크플로를 다른 버전으로 복원할 수 있습니다.

Git을 사용하여 vRealize Orchestrator 콘텐츠 인벤토리를 이전 상태로 재설정

이전 Git 커밋을 사용하면 vRealize Orchestrator 콘텐츠를 이전 상태로 재설정할 수 있습니다.

특정 커밋을 선택하여 vRealize Orchestrator 콘텐츠를 이전 상태로 재설정할 수 있습니다.

사전 요구 사항

- GitHub 또는 GitLab 저장소에 대한 연결을 구성합니다. [Git 저장소에 대한 연결 구성](#)의 내용을 참조하십시오.
- 구성된 Git 저장소로 로컬 변경 집합을 푸시합니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 **관리 > Git 기록**으로 이동합니다.
- 3 재설정할 변경 집합을 선택하고 **이 항목으로 재설정**을 클릭합니다.
- 4 특정 커밋으로 재설정하려는지 확인하고 **확인**을 클릭합니다.

vRealize Orchestrator 콘텐츠 인벤토리가 커밋에 지정된 상태로 재설정됩니다. 관련 vRealize Orchestrator 콘텐츠가 이전 버전으로 되돌려집니다. 커밋이 푸시되었을 때 콘텐츠가 없으면 인벤토리에서 제거됩니다.

다음에 수행할 작업

vRealize Orchestrator 인벤토리를 Git 저장소에 저장된 최신 상태로 복원하려면 **Git 기록** 창에서 Pull 명령을 수행합니다.

vRealize Orchestrator 사용 사례

4

이러한 사용 사례는 vRealize Orchestrator 플랫폼의 기능 일부를 설명합니다.

이러한 사용 사례는 예시 값만 제공합니다. 실제 환경 구조 및 이름 지정 규칙은 다를 수 있습니다.

본 장은 다음 항목을 포함합니다.

- Python을 사용하여 vRealize Orchestrator에서 Amazon Web Services를 통합하는 방법
- Git 분기를 사용하여 내 vRealize Orchestrator 개체 인벤토리를 관리하는 방법
- 타사 모듈을 사용하여 vRealize Automation 프로젝트 API를 호출하는 방법

Python을 사용하여 vRealize Orchestrator에서 Amazon Web Services를 통합하는 방법

이 vRealize Orchestrator 사용 사례는 Python을 사용하여 vRealize Orchestrator 배포의 기능을 확장할 수 있는 방법의 예를 보여 줍니다.

vRealize Orchestrator 8.1부터 작업 및 워크플로 스크립트에서 사용할 수 있는 새로운 런타임 세 가지를 지원합니다.

- Python 3.7
- Node.js 12
- PowerCLI 11/Powershell 6.2
- PowerCLI 12/Powershell 7

참고 PowerCLI 런타임에는 PowerShell을 비롯해 VMware.PowerCLI, PowerNSX 및 PowervRA 모듈이 포함됩니다.

중요 vRealize Orchestrator 배포에서 vRealize Automation 라이선스를 사용하는 경우 새 런타임을 사용할 수 있습니다.

이 사용 사례는 AWS(Amazon Web Services)에서 EC2 인스턴스를 호출하는 Python 스크립트를 생성하는 방법을 보여 줍니다.

중요 사용자 지정 스크립트 개발을 시작하기 전에 vRealize Orchestrator에서 Python, Node.js 및 PowerShell 스크립트를 사용하기 위한 핵심 개념을 숙지하고 있는지 확인합니다. [Python, Node.js 및 PowerShell 스크립트에 대한 핵심 개념](#)의 내용을 참조하십시오.

절차

1 초기 Python 스크립트 생성

로컬 시스템에서 Python 스크립트를 생성하고 스크립트와 boto3 라이브러리를 ZIP 폴더로 패키징합니다.

2 Amazon Web Services 작업 생성

Python 스크립트를 사용하는 vRealize Orchestrator 작업을 생성합니다.

3 Amazon Web Services 작업 디버깅

원래 버전의 Python 스크립트에는 의도적으로 오류가 내장되어 있기 때문에 스크립트를 디버깅하는 방법을 알아볼 수 있습니다.

4 Amazon Web Services 작업 업데이트

업데이트된 Python 스크립트를 가져오고 작업을 다시 실행합니다.

초기 Python 스크립트 생성

로컬 시스템에서 Python 스크립트를 생성하고 스크립트와 boto3 라이브러리를 ZIP 폴더로 패키징합니다.

사전 요구 사항

- Python 3을 다운로드하고 설치합니다. [Python 다운로드 페이지](#)를 참조하십시오.
- Visual Studio 코드를 다운로드하고 설치합니다. [Visual Studio 코드 다운로드 페이지](#)를 참조하십시오.
- Visual Studio 코드용 Python 확장을 설치했는지 확인합니다. [Visual Studio 마켓플레이스](#)를 참조하십시오.

절차

- 1 로컬 시스템에서 vro-python-aws 폴더를 생성하고 여기에 boto3 Python SDK를 설치합니다.

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

- 2 편집기를 열고 기본 Python 스크립트를 생성합니다. 이 사용 사례에서는 Visual Studio 코드를 사용합니다.

```
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)
```

이 Python 스크립트는 지정된 지역에서 실행되고 있는 모든 EC2 인스턴스를 나열합니다.

- 3 생성된 스크립트를 vro-python-aws 폴더에 main.py 파일로 저장합니다.
- 4 명령줄 인터페이스에 로그인합니다.
- 5 vro-python-aws 폴더로 이동합니다.

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

- 6 Python 스크립트가 포함된 ZIP 패키지를 생성합니다.

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

참고 7-Zip과 같은 Zip 유틸리티 도구를 사용하여 ZIP 패키지를 생성할 수도 있습니다.

결과

기본 Python 스크립트를 생성했고 vRealize Orchestrator 배포로 가져올 준비를 마쳤습니다.

Amazon Web Services 작업 생성

Python 스크립트를 사용하는 vRealize Orchestrator 작업을 생성합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 라이브러리 > 작업으로 이동합니다.
- 3 새 작업을 클릭합니다.
- 4 일반 탭에서 작업의 이름, 모듈 및 버전 번호를 입력합니다.
- 5 스크립트 탭에서 런타임으로 Python 3.7을 선택하고 스크립트 유형으로 Zip을 선택합니다.
- 6 가져오기를 클릭합니다.

7 vro-python-aws 폴더로 이동하여 Python 스크립트가 들어 있는 ZIP 패키지를 선택합니다.

8 항목 처리기 텍스트 상자에 **main.handler**를 입력합니다.

참고 작업의 항목 처리기는 가져온 ZIP 패키지의 기본 스크립트를 기반으로 합니다. 기본 스크립트는 **main.py**라는 파일과 **handler**라는 함수에 있으므로 항목 처리기는 **main.handler**여야 합니다. 기본 스크립트 파일의 제목을 다르게 지정한 경우에는 항목 처리기 값을 그에 맞게 변경합니다.

9 작업을 저장하고 **실행**을 클릭합니다.

작업 실행에 오류가 발생합니다.

10 **로그** 탭을 선택합니다.

작업 실행 로그에 **"botocore.exceptions.NoRegionError: 영역을 지정해야 합니다."** 오류 메시지가 표시됩니다. 이는 초기 Python 스크립트에 영역이 정의되지 않아 발생한 예상된 동작입니다.

다음에 수행할 작업

Python 스크립트를 디버깅합니다. [Amazon Web Services 작업 디버깅](#)의 내용을 참조하십시오.

Amazon Web Services 작업 디버깅

원래 버전의 Python 스크립트에는 의도적으로 오류가 내장되어 있기 때문에 스크립트를 디버깅하는 방법을 알아볼 수 있습니다.

사전 요구 사항

AWS(Amazon Web Services) 계정에 로그인하고 이 사용 사례 시나리오를 위한 IAM 사용자를 생성합니다. [AWS 계정에서 IAM 사용자 생성](#)을 참조하십시오. IAM 사용자에게는 다음과 같은 사용 권한이 있어야 합니다.

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

```

절차

1 vRealize Orchestrator Appliance를 준비합니다.

경고 운영 vRealize Orchestrator 배포에서 스크립트를 디버깅하지 마십시오. 개발 및 테스트에 사용하는 단일 노드 vRealize Orchestrator 배포에서 디버깅하십시오.

- a SSH를 통해 vRealize Orchestrator Appliance 명령줄에 **root**로 로그인합니다.
- b `vracli dev tools` 명령을 실행합니다.
- c 계속할지 여부를 확인하는 메시지가 표시됩니다. 계속하려면 **yes**를 입력하고 취소하려면 **no**를 입력합니다.

중요 `vracli dev tools` 명령을 실행하여 Python 스크립트를 디버깅하는 데 필요한 포트를 엽니다. 디버그 프로세스 중에 현재 SSH 세션을 열어 두어야 합니다.

2 디버그 구성을 시작합니다.

- a vRealize Orchestrator 클라이언트에 로그인합니다.
- b AWS 작업을 열고 **디버그**를 클릭합니다.
디버그 프로세스가 시작되고 작업 실행이 일시 중단됩니다.
- c **디버그 구성** 탭을 선택합니다.
탭에는 IDE에 원격으로 연결하여 Python 스크립트를 디버깅할 수 있는 **.json** 구성이 포함되어 있습니다.
- d 구성 콘텐츠를 수동으로 복사하거나 **클립보드에 복사**를 클릭합니다.

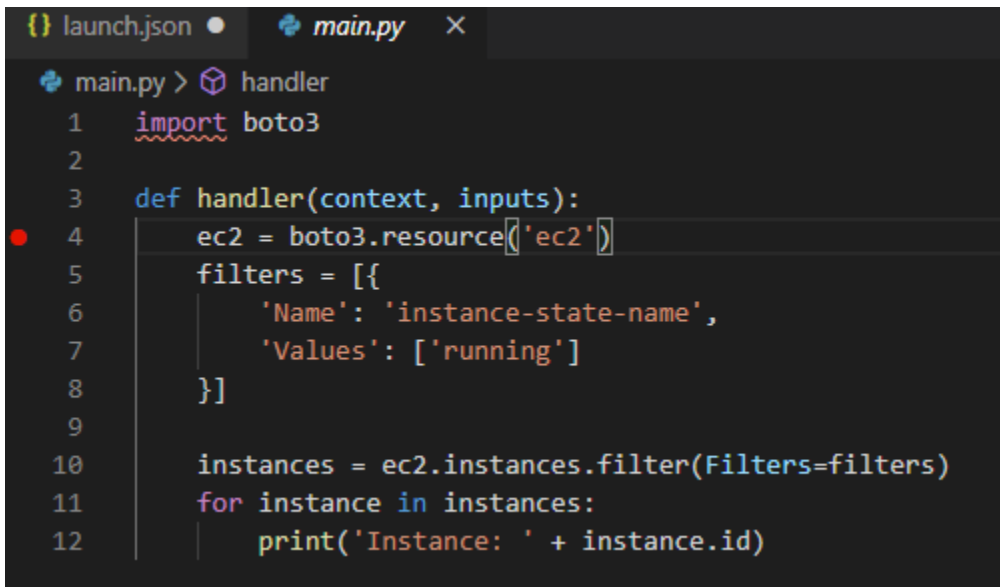
3 Python 스크립트를 디버깅합니다.

- a Visual Studio 코드를 엽니다.
- b `vro-python-aws` 폴더를 엽니다.
- c 상단 탐색 창에서 **실행 > 구성 열기**를 선택합니다.
- d **Python 파일**을 선택합니다.

- e "version" 및 "configuration" 특성을 현재 위치에 그대로 두고 vRealize Orchestrator 클라이언트에서 복사한 .json 구성의 콘텐츠를 붙여넣습니다. 생성된 launch.json 파일은 다음과 비슷해야 합니다.

```
{
  "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 main.py 스크립트 파일을 선택하고 `ec2 = boto3.resource('ec2')` 줄에 중단점을 추가합니다.



```
{ } launch.json ● ● main.py ✕
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 상단 탐색 창에서 **실행 > 디버깅 시작**을 선택합니다.
- h 디버거가 중단점에 도달하면 프로시저 단위 실행 작업을 수행합니다.
- 디버그 실행이 Python 스크립트에 지정된 영역과 AWS 액세스 키가 없음을 나타냅니다.
- i 열려 있는 vRealize Orchestrator Appliance 세션으로 돌아가 **Enter** 키를 눌러 이 디버그 세션에 대해 열었던 포트를 닫습니다.

4 누락된 정보를 Python 스크립트에 추가합니다.

- a Visual Studio 코드에서 Python 스크립트로 ping하려는 AWS 영역 및 IAM 사용자의 AWS 액세스 키가 포함된 **awsconfig**라는 파일을 생성합니다.

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

- b **awsconfig**를 **vro-python-aws** 폴더에 구성(.cfg) 파일로 저장합니다.
- c **main.py** 파일을 열어 **boto3** 라이브러리가 **awsconfig.cfg** 파일을 사용할 수 있도록 수정합니다.

```
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 **main.py** 파일, **awsconfig.cfg** 파일 및 **boto3** 라이브러리가 포함된 새로운 ZIP 패키지를 생성합니다.

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

참고 7-Zip과 같은 Zip 유틸리티 도구를 사용하여 ZIP 패키지를 생성할 수도 있습니다.

Amazon Web Services 작업 업데이트

업데이트된 Python 스크립트를 가져오고 작업을 다시 실행합니다.

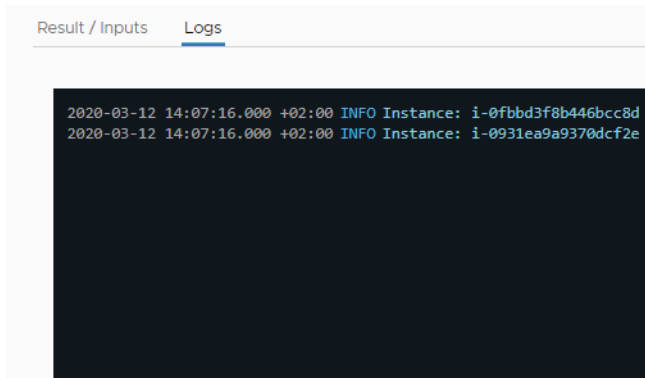
절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리 > 작업**으로 이동한 후 원래 **AWS(Amazon Web Services)** 작업을 선택합니다.
- 3 (선택 사항) **일반** 탭에서 버전 번호를 변경합니다.
- 4 이전 ZIP 패키지를 제거하고 **가져오기**를 클릭합니다.
- 5 업데이트된 ZIP 패키지를 선택합니다.

6 작업을 저장하고 **실행**을 클릭합니다.

7 작업 실행이 완료되면 **로그** 탭을 선택합니다.

로그는 작업에서 쿼리된 EC2 인스턴스를 표시합니다.



다음에 수행할 작업

업데이트된 AWS 작업을 **작업 요소**로 사용하는 vRealize Orchestrator 워크플로를 생성합니다.

Git 분기를 사용하여 내 vRealize Orchestrator 개체 인벤토리를 관리하는 방법

분기를 사용하여 vRealize Orchestrator 콘텐츠가 Git 저장소에서 관리되는 방식을 구성합니다.

Git을 사용하면 중앙 집중식 저장소를 제공하여 vRealize Orchestrator 개발자의 유연성을 높일 수 있습니다. 예를 들어 Git을 사용하여 여러 vRealize Orchestrator 환경에서 워크플로 개발을 관리할 수 있습니다.

참고 Git을 사용하여 개체 인벤토리를 관리하려면 vRealize Orchestrator 배포에서 vRealize Automation 라이선스를 사용해야 합니다. 자세한 내용은 "vRealize Orchestrator 설치 및 구성"에서 "라이선스를 통한 vRealize Orchestrator 기능 사용"을 참조하십시오.

vRealize Orchestrator 8.1부터는 개체를 분기로 푸시하거나 분기에서 개체를 가져올 수 있습니다. 분기를 사용하여 특정 vRealize Orchestrator 개체 그룹이 주 분기로 다시 병합되기 전에 해당 그룹의 개발을 관리할 수 있습니다.

이 사용 사례에서는 GitLab 프로젝트를 사용하여 Python 런타임을 사용하는 vRealize Orchestrator 개체를 관리합니다. 이 사용 사례는 vRealize Orchestrator의 Git 기능에 대한 예를 보여주며 기능 범위의 제한을 나타내지 않습니다.

참고 GitHub에 더 익숙한 경우 이 사용 사례에 GitHub 저장소를 사용할 수 있습니다.

절차

1 GitLab 환경 준비

vRealize Orchestrator Python 개체에 대한 Git 분기를 생성합니다.

2 Git 저장소에 대한 연결 구성

관리자는 vRealize Orchestrator 배포와 Git 저장소 또는 프로젝트 간의 연결을 구성할 수 있습니다.

3 변경 내용을 Git 저장소로 푸시

로컬 vRealize Orchestrator 개체에 대한 변경 내용을 통합 Git 저장소로 푸시합니다. 이 사용 사례의 경우 Python 기반 vRealize Orchestrator 작업에 대한 변경 내용을 특정 Git 분기로 푸시합니다.

GitLab 환경 준비

vRealize Orchestrator Python 개체에 대한 Git 분기를 생성합니다.

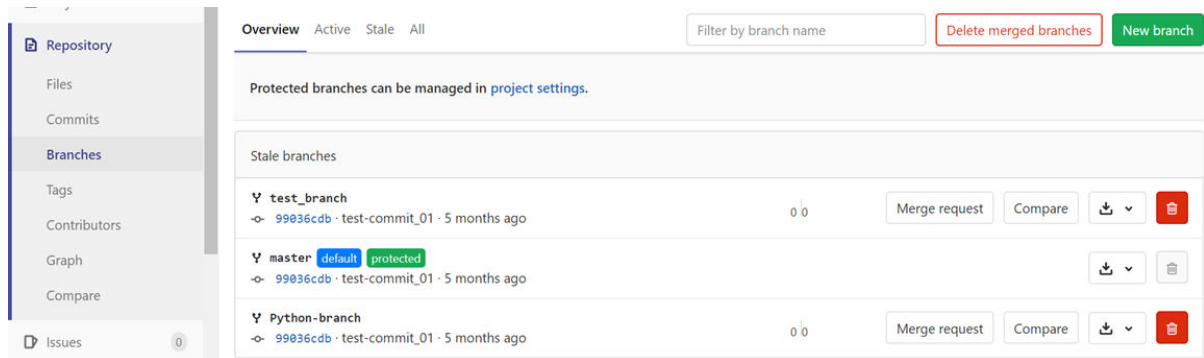
사전 요구 사항

vRealize Orchestrator 환경에 대한 GitLab 프로젝트를 생성합니다. [프로젝트 생성](#)을 참조하십시오.

절차

- 1 GitLab 계정에 로그인합니다.
- 2 GitLab 프로젝트로 이동합니다.
- 3 왼쪽 탐색 창에서 **저장소 > 분기**를 선택합니다.
- 4 개요 탭에서 **새 분기**를 클릭합니다.
- 5 분기 이름에 **Python-branch**를 입력합니다.
- 6 생성 위치 옵션을 **마스터**로 둡니다.
- 7 **분기 생성**을 클릭합니다.

Python 기반 vRealize Orchestrator 개체에 대한 분기를 생성했습니다.



Git 저장소에 대한 연결 구성

관리자는 vRealize Orchestrator 배포와 Git 저장소 또는 프로젝트 간의 연결을 구성할 수 있습니다.

vRealize Orchestrator 개체 인벤토리 관리에 Git을 사용하려면 vRealize Orchestrator Client를 사용하여 Git 저장소에 대한 연결을 구성해야 합니다.

사전 요구 사항

- vRealize Orchestrator 환경에서 vRealize Automation 라이선스를 사용하는지 확인합니다.
- GitLab 프로젝트에 대한 액세스 토큰을 생성하고 구성 과정에서 사용할 수 있도록 클립보드에 복사합니다. [개인 액세스 토큰 생성](#)을 참조하십시오.

참고 이 사용 사례에서는 GitLab 프로젝트를 사용합니다. GitHub에 더 익숙한 경우 GitHub 저장소를 사용할 수 있습니다. GitHub 토큰 생성에 대한 자세한 내용은 [명령줄에 대한 개인 액세스 토큰 생성](#)을 참조하십시오.

절차

- 1 vRealize Orchestrator Client에 **관리자**로 로그인합니다.
- 2 **관리 > Git 저장소**로 이동합니다.
- 3 **저장소 추가**를 클릭합니다.
- 4 Git 저장소의 URL 주소를 입력합니다.

예를 들어 `https://gitlab.com/myusername/my-vro-repo`와 같이 입력합니다.

참고 SSH 프로토콜을 사용하여 연결을 설정할 수도 있습니다.

- 5 Git 프로파일의 사용자 이름을 입력합니다.
- 6 Git 저장소의 액세스 토큰을 입력합니다.
- 7 Git 저장소에 대한 연결을 검증하려면 **검증**을 클릭합니다.
- 8 (선택 사항) vRealize Orchestrator Client에서 저장소를 식별하는 데 사용되는 이름을 변경합니다.
- 9 (선택 사항) 연결된 Git 저장소에 대한 간단한 설명을 추가합니다.
- 10 연결된 Git 저장소를 활성화하려면 **활성 저장소 만들기**를 클릭합니다.

참고 Git 저장소는 한 번에 하나만 활성 상태일 수 있습니다. **Git 저장소** 페이지에서 활성 Git 저장소를 변경할 수 있습니다.

- 11 변경 내용을 푸시할 분기를 선택합니다. 이 사용 사례에서는 **Python 분기**를 사용합니다. [GitLab 환경 준비](#)의 내용을 참조하십시오.

참고 초기 Git 구성을 완료한 후 언제든지 선택한 Git 분기를 변경할 수 있습니다.

- 12 구성 프로세스를 마치하려면 **저장**을 클릭합니다.

다음에 수행할 작업

Git 저장소 메뉴로 다시 돌아가서 저장소의 상태가 **활성**인지 확인합니다.

변경 내용을 Git 저장소로 푸시

로컬 vRealize Orchestrator 개체에 대한 변경 내용을 통합 Git 저장소로 푸시합니다. 이 사용 사례의 경우 Python 기반 vRealize Orchestrator 작업에 대한 변경 내용을 특정 Git 분기로 푸시합니다.

로컬 변경 집합을 Git 저장소로 푸시할 수 있습니다. 각 변경 집합은 하나 이상의 수정된 vRealize Orchestrator 개체로 구성될 수 있습니다.

참고 변경 집합을 Git 저장소로 푸시 및 삭제하는 프로세스는 그룹 사용 권한으로 제한되지 않습니다. 따라서 한 그룹의 워크플로 개발자가 다른 개발자가 변경한 로컬 변경 사항을 푸시 또는 삭제할 수 있습니다.

사전 요구 사항

- Git 분기를 생성했는지 확인합니다. [GitLab 환경 준비](#)의 내용을 참조하십시오.
- Git 저장소와의 연결을 구성했는지 확인합니다. [Git 저장소에 대한 연결 구성](#)의 내용을 참조하십시오.
- Git 통합이 변경 내용을 **Python 분기** Git 분기로 푸시하도록 설정되어 있는지 확인합니다.
- Python 기반 vRealize Orchestrator 개체를 생성합니다. 예를 들어 [Python을 사용하여 vRealize Orchestrator에서 Amazon Web Services를 통합하는 방법](#)의 내용을 참조하십시오.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 Python 작업을 편집합니다.
 - a **라이브러리 > 작업**으로 이동하고 Python 작업을 선택합니다.
 - b 작업과 관련된 일부 내용을 변경합니다(예: 설명 변경).
 - c 작업을 저장합니다.

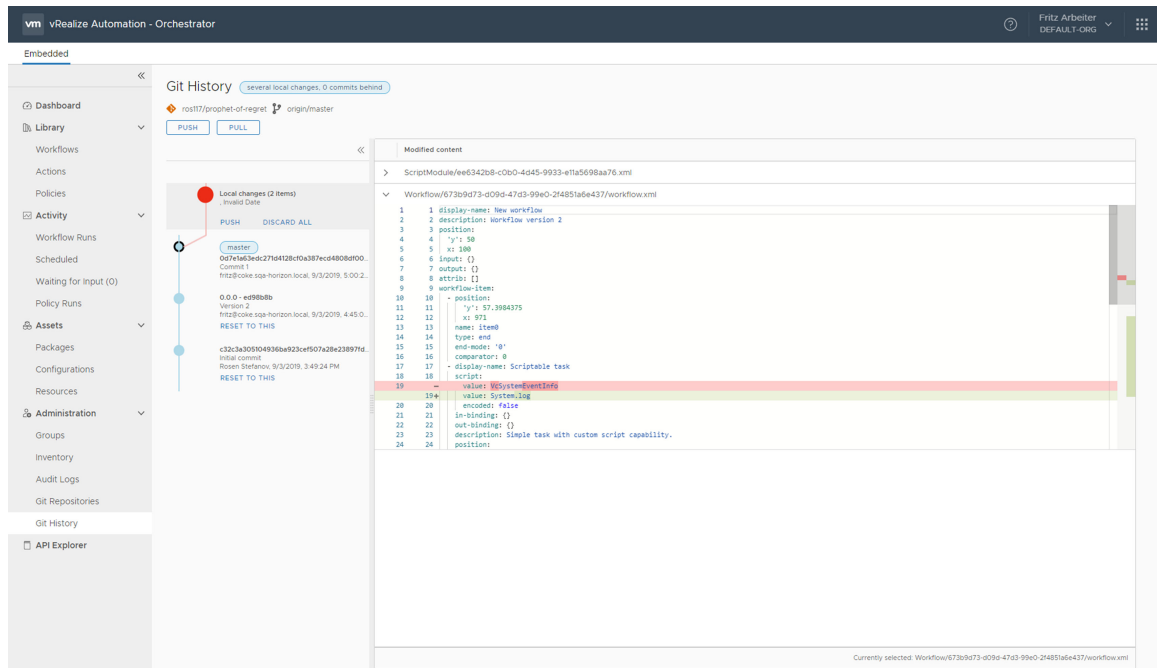
3 변경 내용을 Git 저장소로 푸시합니다.

참고 개체 편집기 맨 아래에 표시된 **버전** 옵션을 클릭하여 개체 수준별로 로컬 변경 내용을 푸시할 수도 있습니다.

a 관리 > Git 기록으로 이동합니다.

Git 기록에는 로컬 버전 분기와 선택된 **Git** 저장소 분기 사이의 현재 차이점이 표시됩니다. 수정된 vRealize Orchestrator 개체에 대한 항목을 확장하여 버전 차이를 확인할 수 있습니다.

참고 모두 삭제를 선택하여 로컬 변경 집합을 삭제할 수 있습니다.



b 푸시를 클릭합니다.

c 커밋 제목을 입력합니다.

d (선택 사항) 커밋에 대한 간략한 설명을 입력합니다.

e Git 저장소로 푸시할 Python 작업에 대한 변경 내용을 선택합니다.

4 로컬 변경 집합의 Git 저장소 푸시를 완료하려면 푸시를 클릭합니다.

다음에 수행할 작업

Git 분기에서 개발을 완료한 후 주 분기와 병합합니다. **병합 요청을 생성하는 방법**을 참조하십시오.

타사 모듈을 사용하여 vRealize Automation 프로젝트 API를 호출하는 방법

이 vRealize Orchestrator 사용 사례에서는 타사 모듈을 사용하여 vRealize Automation 프로젝트 API를 호출하는 방법을 보여줍니다.

vRealize Orchestrator 8.1부터는 다음 런타임을 사용할 수 있습니다.

- Python 3.7
- Node.js 12
- PowerCLI 11/Powershell 6.2
- PowerCLI 12/Powershell 7

참고 PowerCLI 런타임에는 PowerShell을 비롯해 VMware.PowerCLI, PowerNSX 및 PowervRA 모듈이 포함됩니다.

이 사용 사례에서는 타사 종속성 모듈을 사용하여 vRealize Automation 프로젝트 API에 연결하는 vRealize Orchestrator 작업을 생성하는 방법을 알아봅니다.

중요 사용자 지정 스크립트 개발을 시작하기 전에 vRealize Orchestrator에서 Python, Node.js 및 PowerShell 스크립트를 사용하기 위한 핵심 개념을 숙지하고 있는지 확인합니다. [Python, Node.js 및 PowerShell 스크립트에 대한 핵심 개념](#)의 내용을 참조하십시오.

vRealize Automation 프로젝트 API를 호출하는 Python 스크립트 생성

Python을 사용하여 vRealize Automation 프로젝트 API를 호출하는 샘플 스크립트를 생성합니다.

사전 요구 사항

Python 3 및 PIP 패키지 설치 관리자를 설치했는지 확인합니다. [Python 다운로드 페이지](#) 및 [Python 패키지 인덱스](#)를 참조하십시오.

절차

- 1 로컬 시스템에서 명령줄 셸을 엽니다.
- 2 vro-python-vra 폴더를 생성합니다.

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

- 3 vro-python-vra 폴더로 이동합니다.

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

- 4 handler.py라는 Python 스크립트를 생성합니다.

```
touch handler.py
```

`handler.py` 스크립트는 두 개의 인수(vRealize Orchestrator 워크플로 실행의 컨텍스트 및 바운드 vRealize Orchestrator 입력)를 받아들이는 하나의 함수를 정의해야 합니다.

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

참고 표준 로깅 라이브러리를 사용하면 스크립트를 사용하는 작업에 로깅한 모든 내용이 워크플로 로그에도 표시됩니다. 스크립트의 입력 및 반환에는 vRealize Orchestrator Client에 구성된 해당 입력 매개 변수와 반환 유형이 있어야 합니다. 예를 들어, 스크립트에 vRAUr1 입력이 있으면 vRealize Orchestrator Client에 vRAUr1이라는 해당 입력 매개 변수가 있어야 합니다. 마찬가지로 스크립트가 문자열 값을 반환하는 경우에는 vRealize Orchestrator 클라이언트에 구성된 반환 유형도 문자열 유형이어야 합니다. 작업에서 복합 개체를 반환하는 경우에는 **Properties** 또는 **Composite Type** 반환 유형을 사용할 수 있습니다.

5 Python 요청 모듈을 설치합니다.

중요 타사 종속성 모듈은 기본 `vro-python-vra` 스크립트 폴더의 루트 수준 폴더에 설치해야 합니다. 이 사용 사례에서는 요청 모듈에 대한 `lib` 폴더를 생성합니다.

a lib 폴더를 생성합니다.

```
mkdir lib
```

b 요청 모듈을 설치합니다.

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

6 요청 모듈을 `handler.py` 스크립트에 추가합니다.

```
import requests

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

7 vRealize Automation 프로젝트 API에 대한 GET 요청을 생성합니다.

```
token = ''
vRAUr1 = ''
r = requests.get(vRAUr1 + '/iaas/api/projects', headers={'Authorization': 'Bearer ' + token})

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


8 token 및 vRAUrl 값을 정의합니다.

- a vRealize Automation ID 서비스 API를 사용하여 액세스 토큰을 검색합니다. [vRealize Automation API에 대한 액세스 토큰 가져오기](#)를 참조하십시오.
- b vRAUrl 값의 경우, 이름이 같은 vRealize Orchestrator 입력 매개 변수를 사용하도록 스크립트를 정의합니다.

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

- c handler.py 파일에 새 값을 추가합니다.

```
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()
```

참고 vRealize Automation 프로젝트 API의 응답이 JSON 형식으로 반환되므로 vRealize Orchestrator 작업에 Properties 또는 Composite Type 반환 유형을 사용합니다.

9 요청 모듈의 lib 폴더와 handler.py 파일이 포함된 ZIP 패키지를 생성합니다.

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

다음에 수행할 작업

PowerShell 스크립트를 vRealize Orchestrator 작업으로 가져옵니다. [vRealize Orchestrator 클라이언트에서 작업 생성](#)의 내용을 참조하십시오.

vRealize Automation 프로젝트 API를 호출하는 Node.js 스크립트 생성

Node.js를 사용하여 vRealize Automation 프로젝트 API를 호출하는 샘플 스크립트를 생성합니다.

사전 요구 사항

Node.js 12를 다운로드하고 설치합니다. [Node.js 다운로드](#)를 참조하십시오.

절차

- 1 로컬 시스템에서 명령줄 셸을 엽니다.
- 2 vro-node-vra 폴더를 생성합니다.

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

3 vro-node-vra 폴더로 이동합니다.

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

4 handler.js라는 Node.js 스크립트를 생성합니다.

```
touch handler.js
```

handler.js 스크립트는 두 개의 인수(vRealize Orchestrator 워크플로 실행의 컨텍스트 및 바운드 vRealize Orchestrator 입력)를 받아들이는 하나의 함수를 정의해야 합니다.

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

참고 표준 로깅 라이브러리를 사용하면 스크립트를 사용하는 작업에 로깅한 모든 내용이 워크플로 로그에도 표시됩니다. 스크립트의 입력 및 반환에는 vRealize Orchestrator Client에 구성된 해당 입력 매개 변수와 반환 유형이 있어야 합니다. 예를 들어, 스크립트에 vRAUrl 입력이 있으면 vRealize Orchestrator Client에 vRAUrl이라는 해당 입력 매개 변수가 있어야 합니다. 마찬가지로 스크립트가 문자열 값을 반환하는 경우에는 vRealize Orchestrator 클라이언트에 구성된 반환 유형도 문자열 유형이어야 합니다. 작업에서 복합 개체를 반환하는 경우에는 Properties 또는 Composite Type 반환 유형을 사용할 수 있습니다.

5 Node.js 요청 모듈을 설치합니다.

```
npm install request
```

중요 타사 종속성 모듈은 기본 vro-node-vra 스크립트 폴더의 루트 수준 node_modules 폴더에 설치해야 합니다. 이 폴더를 옮기거나 이름을 변경하지 마십시오.

6 요청 모듈을 handler.js 스크립트에 추가합니다.

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

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

7 vRealize Automation 프로젝트 API에 대한 GET 요청을 생성합니다.

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

8 token 및 vRAUrl 값을 정의합니다.

- a vRealize Automation ID 서비스 API를 사용하여 액세스 토큰을 검색합니다. [vRealize Automation API에 대한 액세스 토큰 가져오기](#)를 참조하십시오.
- b vRAUrl 값의 경우, 이름이 같은 vRealize Orchestrator 입력 매개 변수를 사용하도록 스크립트를 정의합니다.

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

- c handler.js 파일에 새 값을 추가합니다.

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

참고 vRealize Automation 프로젝트 API의 응답이 JSON 형식으로 반환되므로 vRealize Orchestrator 작업에 Properties 또는 Composite Type 반환 유형을 사용합니다.

9 요청 모듈의 node_modules 폴더와 handler.js 파일이 포함된 ZIP 패키지를 생성합니다.

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

다음에 수행할 작업

Node.js 스크립트를 vRealize Orchestrator 작업으로 가져옵니다. [vRealize Orchestrator 클라이언트에 서 작업 생성](#)의 내용을 참조하십시오.

vRealize Automation 프로젝트 API를 호출하는 PowerShell 스크립트 생성

PowerShell을 사용하여 vRealize Automation 프로젝트 API를 호출하는 샘플 스크립트를 생성합니다.

절차

- 1 로컬 시스템에서 명령줄 셸을 엽니다.
- 2 vro-powershell-vra 폴더를 생성합니다.

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

- 3 vro-powershell-vra 폴더로 이동합니다.

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

4 handler.ps1이라는 PowerShell 스크립트를 생성합니다.

```
touch handler.ps1
```

handler.ps1 스크립트는 두 개의 인수(vRealize Orchestrator 워크플로 실행의 컨텍스트 및 바운드 vRealize Orchestrator 입력)를 받아들이는 하나의 함수를 정의해야 합니다.

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

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

참고 표준 로깅 라이브러리를 사용하면 스크립트를 사용하는 작업에 로깅한 모든 내용이 워크플로 로그에도 표시됩니다. 스크립트의 입력 및 반환에는 vRealize Orchestrator Client에 구성된 해당 입력 매개 변수와 반환 유형이 있어야 합니다. 예를 들어, 스크립트에 vRAur1 입력이 있으면 vRealize Orchestrator Client에 vRAur1이라는 해당 입력 매개 변수가 있어야 합니다. 마찬가지로 스크립트가 문자열 값을 반환하는 경우에는 vRealize Orchestrator 클라이언트에 구성된 반환 유형도 문자열 유형이어야 합니다. 작업에서 복합 개체를 반환하는 경우에는 **Properties** 또는 **Composite Type** 반환 유형을 사용할 수 있습니다.

5 PowerShell 어설션 모듈을 설치합니다.

중요 타사 종속성 모듈은 기본 vro-powershell-vra 스크립트 폴더의 루트 수준 폴더에 설치해야 합니다. 이 사용 사례에서는 어설션 모듈에 대한 **Modules** 폴더를 생성합니다.

a Modules 폴더를 생성합니다.

```
mkdir Modules
```

b 어설션 모듈을 설치합니다.

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

6 어설션 모듈을 handler.ps1 스크립트에 추가합니다.

```
Import-Module Assert

function Handler {
    Param($context, $inputs)

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

- 7 Invoke-RestMethod cmdlet을 사용하는 vRealize Automation 프로젝트 API에 대한 GET 요청을 생성합니다.

```
$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 token 및 vRAUrl 값을 정의합니다.

- a vRealize Automation ID 서비스 API를 사용하여 액세스 토큰을 검색합니다. [vRealize Automation API에 대한 액세스 토큰 가져오기](#)를 참조하십시오.
- b Assert-NotNull 및 Assert-Type 어설션 모듈 특성을 추가합니다.

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

- c vRAUrl 값의 경우, 이름이 같은 vRealize Orchestrator 입력 매개 변수를 사용하도록 스크립트를 정의합니다.

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

- d handler.ps1 파일에 새 값을 추가합니다.

```
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
}
```

참고 vRealize Automation 프로젝트 API의 응답이 JSON 형식으로 반환되므로 vRealize Orchestrator 작업에 Properties 또는 Composite Type 반환 유형을 사용합니다.

9 어설션 모듈의 **Modules** 폴더와 **handler.ps1** 파일이 포함된 ZIP 패키지를 생성합니다.

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

다음에 수행할 작업

PowerShell 스크립트를 vRealize Orchestrator 작업으로 가져옵니다. [vRealize Orchestrator 클라이언트에서 작업 생성](#)의 내용을 참조하십시오.

워크플로 관리

5

워크플로는 순차적으로 실행되는 일련의 작업 및 의사 결정입니다. vRealize Orchestrator는 일반적인 관리 작업을 수행하는 워크플로의 라이브러리를 제공합니다. 또한 vRealize Orchestrator는 워크플로가 수행하는 개별 작업의 라이브러리를 제공합니다.

워크플로는 특정 순서로 수행될 때 가상 환경에서 특정 작업이나 특정 프로세스를 완료하는 작업, 결정 및 결과를 조합한 것입니다. 워크플로는 가상 시스템 프로비저닝, 백업, 정기 유지 보수 수행, 이메일 전송, SSH 작업 수행, 물리적 인프라 관리 및 기타 일반 유틸리티 작업 등의 작업을 수행합니다. 워크플로는 그 기능에 따라 입력을 허용합니다. 사용자는 정의된 스케줄에 따라 실행되거나 특정 예상 이벤트가 발생하면 실행되는 워크플로를 생성할 수 있습니다. 정보는 사용자 본인, 다른 사용자, 다른 워크플로, 작업 또는 애플리케이션이 보낸 웹 서비스 호출 등 외부 프로세스가 제공할 수 있습니다. 워크플로는 실행 전 정보의 유효성 검사 및 필터링을 어느 정도 수행합니다.

워크플로는 다른 워크플로를 호출할 수 있습니다. 예를 들어 새 가상 시스템을 생성하기 위해 다른 워크플로를 호출하는 워크플로를 가질 수 있습니다.

사용자는 워크플로 라이브러리에 액세스하고 워크플로 엔진에서 워크플로를 실행할 수 있는 vRealize Orchestrator Client 인터페이스의 IDE(통합 개발 환경)를 사용하여 워크플로를 생성할 수 있습니다. 워크플로 엔진은 또한 vRealize Orchestrator에 플러그인한 외부 라이브러리에서 개체를 가져올 수도 있습니다. 이 기능을 사용하면 프로세스를 사용자 지정하거나 타사 애플리케이션이 제공하는 기능을 구현할 수 있습니다.

본 장은 다음 항목을 포함합니다.

- vRealize Orchestrator 워크플로 라이브러리의 표준 워크플로
- vRealize Orchestrator 클라이언트에서 워크플로 생성
- 상위 워크플로에서 워크플로 및 작업 편집
- vRealize Orchestrator 입력 양식 디자이너
- vRealize Orchestrator 클라이언트에서 사용자 상호 작용 요청
- vRealize Orchestrator 클라이언트에서 워크플로 예약

vRealize Orchestrator 워크플로 라이브러리의 표준 워크플로

vRealize Orchestrator에는 가상 인프라에서 작업을 자동화하는 데 사용할 수 있는 표준 워크플로 라이브러리가 제공됩니다. 표준 라이브러리의 워크플로는 읽기 전용 상태로 잠겨 있습니다. 표준 워크플로를 사용자 지정하려면 해당 워크플로를 복제해야 합니다. 복제한 워크플로나 사용자가 만든 사용자 지정 워크플로는 모두 편집할 수 있습니다.

워크플로 라이브러리의 콘텐츠는 HTML5 기반 vRealize Orchestrator Client의 **라이브러리 > 워크플로** 메뉴를 통해 액세스할 수 있습니다. 클라이언트의 표준 및 사용자 지정 워크플로는 모두 태그를 사용하여 구성됩니다. 예를 들어 워크플로 라이브러리 검색 상자에 **SSH**를 입력하여 **키 쌍 생성** 워크플로에 액세스할 수 있습니다.

참고 워크플로를 복제하는 경우가 아니면 표준 워크플로에 새 태그를 추가할 수 없습니다.

vRealize Orchestrator 클라이언트에서 워크플로 생성

vRealize Orchestrator Client를 사용하여 워크플로를 생성하고 편집할 수 있습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리 > 워크플로**를 선택합니다.
- 3 **새 워크플로**를 클릭합니다.
- 4 새 워크플로의 이름을 입력하고 **생성**을 클릭합니다.
- 5 워크플로 편집기를 사용하여 변수, 워크플로 입력 및 출력, 스키마 구조 및 워크플로의 프레젠테이션을 구성합니다.
- 6 워크플로 편집을 완료하려면 **저장**을 클릭합니다.

참고 **버전 기록** 탭에서 워크플로의 변경 내용을 추적할 수 있습니다. 자세한 내용은 [vRealize Orchestrator 개체 버전 기록](#) 항목을 참조하십시오.

다음에 수행할 작업

vRealize Orchestrator 토큰 재생 기능을 사용하여 워크플로의 성능을 최적화할 수 있습니다. 자세한 내용은 [vRealize Orchestrator Client에서 워크플로 토큰 재생 사용](#) 항목을 참조하십시오.

상위 워크플로에서 워크플로 및 작업 편집

vRealize Orchestrator Client의 상위 워크플로에서 직접 워크플로 및 작업을 편집합니다.

상위 워크플로에서 하위 워크플로 및 작업을 직접 편집하면 워크플로 개발을 간소화하는 데 도움이 될 수 있습니다.

사전 요구 사항

다른 워크플로, 작업 또는 둘 다를 호출하는 워크플로를 생성합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리 > 워크플로**로 이동하여 워크플로를 선택합니다.
- 3 **스키마** 탭을 선택합니다.
- 4 개체 유형에 따라 워크플로 캔버스에서 **워크플로 요소** 또는 **작업 요소**를 두 번 클릭합니다.
- 5 개체를 편집합니다.
- 6 하위 워크플로 또는 작업 편집을 완료하려면 **저장**을 클릭합니다.
- 7 상위 워크플로로 돌아가려면 개체 편집기를 닫습니다.

vRealize Orchestrator 입력 양식 디자이너

워크플로에 입력 매개 변수가 필요하면 사용자가 필요한 값을 입력하는 대화상자가 열립니다. 입력 양식 디자이너를 사용하여 이 대화상자의 콘텐츠, 레이아웃 및 프레젠테이션을 구성할 수 있습니다.

입력 양식 디자이너는 워크플로 편집기의 **입력 양식** 탭에 있습니다. 이 디자이너는 탐색 메뉴, 디자인 캔버스 및 속성 메뉴로 구성됩니다. 입력 및 일반 요소를 왼쪽 메뉴에서 설계 캔버스로 끌어올 수 있습니다. 캔버스에서 입력 매개 변수의 위치를 설정하고, 별도의 입력 탭으로 구성하고, 입력 매개 변수 속성을 구성할 수 있습니다.

참고 입력 양식 디자이너에서 워크플로 편집기의 **변수** 탭에 있는 콘텐츠는 사용할 수 없습니다. **입력/출력** 탭의 매개 변수만 사용할 수 있습니다.

일반 요소

드롭다운 메뉴 및 암호 텍스트 상자과 같은 일반 요소를 입력 양식 디자이너에 추가할 수 있습니다. 일반 요소는 실제 입력 매개 변수와는 일치하지 않지만 입력 매개 변수에 바인딩할 수 있습니다.

vRealize Orchestrator 클라이언트에서 워크플로 입력 매개 변수 대화 상자 생성

입력 양식 디자이너를 사용하여 워크플로 입력 매개 변수 대화상자를 생성하고 사용자 지정할 수 있습니다.

사전 요구 사항

워크플로에 정의된 입력 매개 변수 목록이 있는지 확인합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.

2 라이브러리 > 워크플로로 이동합니다.

3 사용자 지정 워크플로를 선택합니다.

4 입력 양식 탭을 클릭합니다.

5 (선택 사항) 입력 대화상자에서 사용할 탭을 생성합니다.

탭을 사용하여 대화상자의 구조를 구성할 수 있습니다.

6 입력 매개 변수를 선택합니다.

7 입력 매개 변수의 속성을 편집합니다.

입력 매개 변수 속성에 대한 자세한 내용은 [vRealize Orchestrator 클라이언트의 입력 매개 변수 속성 항목을 참조하십시오.](#)

8 (선택 사항) 캔버스에 일반 요소를 추가하여 입력 매개 변수에 바인딩합니다.

9 (선택 사항) 입력 매개 변수에 외부 검증을 추가합니다. 자세한 내용은 [작업을 사용하여 vRealize Orchestrator 워크플로 입력 검증](#) 항목을 참조하십시오.

10 저장을 클릭합니다.

결과

워크플로 대화상자의 레이아웃이 생성되고 입력 매개 변수의 속성이 설정되었습니다.

vRealize Orchestrator 클라이언트의 입력 매개 변수 속성

매개 변수 속성을 설정하여 사용자가 vRealize Orchestrator 워크플로를 실행할 때 입력하는 입력 매개 변수를 제한할 수 있습니다.

vRealize Orchestrator에서는 워크플로에 사용되는 입력 매개 변수 값을 정량화하는 데 사용되는 매개 변수 속성을 정의할 수 있습니다. 정의한 매개 변수 속성은 사용자가 vRealize Orchestrator 워크플로에 제공할 수 있는 입력 매개 변수의 유형과 값을 제한합니다.

매개 변수 속성은 입력 매개 변수의 유효성을 검사하고 입력 매개 변수 대화 상자에 텍스트 상자가 나타나는 방식을 수정합니다. 일부 매개 변수 속성은 매개 변수 간 종속성을 생성할 수 있습니다.

매개 변수 속성	설명
레이블	입력 매개 변수 레이블을 설정합니다.
표시 유형	입력 텍스트 상자 표시 유형을 설정합니다.
가시성	입력 매개 변수의 표시 여부를 설정합니다.
읽기 전용	입력 텍스트 상자를 읽기 전용으로 설정합니다.
사용자 지정 도움말	입력 매개 변수 표시판 설명을 설정합니다.
기본값	입력 매개 변수의 기본값을 설정합니다.
단계	숫자 유형 입력에 사용됩니다. 클릭당 입력 매개 변수의 값이 증가하는 정도를 설정합니다.
필수	입력 매개 변수 값이 필수인지 여부를 설정합니다.

매개 변수 속성	설명
정규식	정규식을 사용하여 입력의 유효성을 검사합니다.
최소값	매개 변수의 최소값 또는 최소 길이를 설정합니다.
최대값	매개 변수의 최대값 또는 최대 길이를 설정합니다.
텍스트 상자 일치	입력 매개 변수 값을 다른 입력 매개 변수의 값과 일치하도록 설정합니다.
값 소스	<p>화면 표시, 값 및 제약 조건 탭에서 매개 변수 속성의 값 소스를 설정합니다.</p> <p>참고 외부 소스를 사용하여 외부 작업의 값을 가져올 수 있습니다. 사용 가능한 작업의 필터링은 매개 변수 유형에 의해 수행됩니다.</p>

작업을 사용하여 vRealize Orchestrator 워크플로 입력 검증

외부 작업을 사용하여 사용자 지정 워크플로의 입력을 검증합니다.

사전 요구 사항

입력 매개 변수를 사용하여 사용자 지정 워크플로를 생성합니다. 자세한 내용은 [vRealize Orchestrator 클라이언트에서 워크플로 생성](#) 항목을 참조하십시오.

입력 양식 디자이너를 사용하여 워크플로 입력에 대한 외부 검증을 생성할 수 있습니다. 외부 검증은 입력 매개 변수 값에 오류가 포함된 경우 문자열 값을 반환하는 작업 스크립트를 사용합니다. 입력 매개 변수 값이 유효하면 외부 검증은 아무 것도 반환하지 않습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 검증 작업을 생성합니다.
 - a 라이브러리 > 작업으로 이동합니다.
 - b 새 작업을 클릭합니다.
 - c 요약 탭에 필요한 정보를 입력합니다.
 - d 검증 작업 입력 매개 변수를 입력합니다.

참고 검증 작업 입력 매개 변수의 이름은 검증되는 워크플로 입력 매개 변수의 이름과 동일해야 합니다.

- e **스크립트** 탭에 검증 작업의 스크립트를 입력합니다.

```
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
```

참고 위의 스크립트는 간단한 예제이며 사용할 수 있는 검증 스크립트의 전체 범위를 나타내지는 않습니다.

- f **저장**을 클릭합니다.
- 3 외부 검증을 적용합니다.
- a **라이브러리 > 워크플로**로 이동합니다.
- b 사용자 지정 워크플로를 선택합니다.
- c **입력 양식** 탭을 선택합니다.
- d 화면 왼쪽 위에서 클립보드 아이콘을 선택합니다.
- e vRealize Orchestrator 검증 요소를 캔버스로 끌어옵니다.
- f 검증 요소를 선택하고 검증 레이블을 입력한 다음 검증 작업을 선택합니다.
- g (선택 사항) 추가 검증 요소를 생성합니다.
- h **저장**을 클릭합니다.

- 4 워크플로를 실행합니다.

검증에서 오류가 발생하면 문자열이 반환됩니다. 검증이 성공하면 검증에서 아무 것도 반환되지 않고 워크플로가 계속 실행됩니다.

결과

사용자 지정 vRealize Orchestrator 워크플로에 대한 외부 검증이 생성되었습니다.

vRealize Orchestrator 클라이언트에서 사용자 상호 작용 요청

워크플로가 완료되기 전에 워크플로에서 추가 사용자 입력을 요청할 수 있습니다.

추가 사용자 상호 작용이 필요한 워크플로는 사용자가 요청한 입력 매개 변수를 입력할 때까지 작업을 일시 중단합니다. 워크플로는 요청된 정보를 제공할 수 있는 사용자를 정의하고 이에 따라 상호 작용 요청을 보냅니다. 사용자 입력을 기다리는 워크플로는 vRealize Orchestrator Client 대시보드 및 오른쪽 상단 알림 메뉴의 **최근 워크플로 실행** 패널에 표시됩니다.

vRealize Orchestrator 클라이언트에서 워크플로 예약

vRealize Orchestrator 워크플로 실행을 자동화하기 위해 예약을 사용할 수 있습니다.

워크플로 실행을 예약할 때 예약된 작업을 실행하는 날짜, 시간 및 간격을 설정합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리** 메뉴에서 워크플로를 선택하고 워크플로 패널에서 **스케줄**을 클릭합니다.
- 3 **일반**, **스케줄링** 및 **워크플로** 탭에서 예약된 작업 매개 변수를 구성합니다.

참고 워크플로 매개 변수 범주는 입력 매개 변수가 필요한 워크플로에 대해서만 표시됩니다.

매개 변수	설명
이름	예약된 작업의 이름입니다.
설명	예약된 작업의 용도를 자세히 보여주는 간단한 설명입니다.
시작	워크플로의 첫 번째 예약된 실행의 날짜 및 시간입니다.
과거인 경우 시작	예약된 시간이 과거인 경우 워크플로를 시작할지를 선택합니다. 예 는 예약된 워크플로를 즉시 시작합니다. 아니요 는 다음에 예약된 되풀이 시 워크플로를 시작합니다.
스케줄	예약된 작업의 되풀이 패턴 및 이벤트 트리거 항목을 설정합니다.
종료 날짜	되풀이 없음 이 선택된 경우에만 표시됩니다. 예약된 작업을 종료할 날짜 및 시간을 설정합니다.
워크플로	워크플로의 입력 매개 변수를 입력합니다.

- 4 **생성**을 클릭합니다.

결과

이렇게 하여 워크플로의 예약된 작업을 생성할 수 있습니다. 예약된 워크플로는 **작업 > 예약됨** 아래에 나타납니다. 스케줄 패널에서 **삭제**를 클릭하여 예약된 작업을 삭제할 수 있습니다.

vRealize Orchestrator 클라이언트에서 예약된 작업 편집

예약된 워크플로의 날짜, 시간 및 되풀이 같은 매개 변수를 변경하기 위해 예약된 작업을 편집할 수 있습니다.

사전 요구 사항

예약된 워크플로 작업을 생성합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.

- 2 **작업 > 예약됨**에서 예약된 작업을 선택합니다.
- 3 워크플로 패널에서 **편집**을 클릭합니다.
- 4 스케줄을 편집하고 **저장**을 클릭합니다.

참고 예약된 작업을 생성할 때 설정하는 입력 매개 변수는 읽기 전용이며 편집할 수 없습니다. 이러한 매개 변수를 변경하려면 이 워크플로에서 새로운 예약된 작업을 생성합니다.

작업 관리

6

작업 스크립트를 추가하여 vRealize Orchestrator 워크플로를 수정할 수 있습니다.

vRealize Orchestrator Client에는 미리 정의된 작업 라이브러리와 사용자 지정 작업 스크립트에 대한 작업 편집기가 제공됩니다. 작업은 워크플로에서 빌딩 블록으로 사용하는 개별 함수를 나타냅니다.

작업은 JavaScript 함수로서, 여러 입력 매개 변수를 사용하고 단일 반환 값을 제공할 수 있습니다.

vRealize Orchestrator API의 모든 개체 또는 플러그인을 사용하여 vRealize Orchestrator로 가져오는 모든 API의 개체를 호출할 수 있습니다.

워크플로가 실행되면 작업은 워크플로의 변수에서 입력 매개 변수를 가져옵니다. 이러한 변수는 워크플로의 초기 입력 매개 변수이거나 워크플로가 실행될 때 워크플로의 다른 요소가 설정하는 변수일 수 있습니다.

작업 편집기에는 스크립트에 대한 자동 완성 기능과 사용 가능한 스크립팅 유형 및 해당 설명서가 포함된 API 탐색기가 포함되어 있습니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트에서 작업 생성](#)
- [작업 실행 및 디버깅](#)
- [Python, Node.js 및 PowerShell 스크립트에 대한 핵심 개념](#)
- [Python, Node.js 및 PowerShell 스크립트에 대한 런타임 제한](#)

vRealize Orchestrator 클라이언트에서 작업 생성

vRealize Orchestrator Client를 사용하여 작업 스크립트를 생성, 편집 및 삭제할 수 있습니다.

vRealize Orchestrator 8.1부터는 다음 런타임을 사용할 수 있습니다.

- Python 3.7
- Node.js 12
- PowerCLI 11/Powershell 6.2

■ PowerCLI 12/Powershell 7

참고 PowerCLI 런타임에는 PowerShell을 비롯해 VMware.PowerCLI, PowerNSX 및 PowervRA 모듈이 포함됩니다.

사전 요구 사항

Python, Node.js 또는 PowerShell 스크립트를 생성하기 전에 이러한 런타임을 사용하는 vRealize Orchestrator 호환 스크립트를 개발하기 위한 핵심 개념을 숙지하고 있는지 확인합니다. [Python](#), [Node.js](#) 및 [PowerShell 스크립트에 대한 핵심 개념](#)의 내용을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **라이브러리 > 작업**으로 이동합니다.
- 3 **새 작업**을 클릭합니다.
- 4 **일반** 탭에서 작업의 이름과 모듈 이름을 입력합니다.

참고 이 이름과 모듈 이름은 모든 작업에 대해 고유해야 합니다. 작업 이름은 올바른 JavaScript 함수여야 합니다. 작업 이름은 단일 단어여야 하며 문자, 숫자, 달러("\$") 및 밑줄("_") 기호만 포함할 수 있습니다. 모듈 이름은 점(".") 문자로 구분된 단어로 구성되어야 합니다.

- 5 (선택 사항) 작업에 대한 설명, 버전 번호, 태그 및 그룹 사용 권한을 생성합니다.
- 6 **스크립트** 탭에서 작업 입력을 추가하고 출력의 반환 유형을 선택한 다음, 스크립트를 작성합니다.

참고 **유형** 드롭다운 메뉴에서 **Zip**을 선택하면 외부 스크립트 소스와 종속성 모듈(해당하는 경우)을 가져올 수 있습니다.

- 7 작업 편집을 완료하려면 **저장**을 클릭합니다.

작업이 저장되었다는 메시지가 표시됩니다.

다음에 수행할 작업

vRealize Orchestrator 작업을 사용하는 방법에 대한 사용 사례를 보려면 [Python을 사용하여 vRealize Orchestrator에서 Amazon Web Services를 통합하는 방법](#) 항목을 참조하십시오.

작업 실행 및 디버깅

작업 편집기에서 직접 작업을 실행하고 디버깅하여 작업을 개선할 수 있습니다.

vRealize Orchestrator 8.1부터 vRealize Orchestrator 클라이언트의 작업 편집기에서 바로 작업을 실행하고 디버깅할 수 있습니다. 이 기능을 사용하면 워크플로에 통합될 때 작업이 예상대로 수행되도록 보장할 수 있습니다.

vRealize Orchestrator 클라이언트에서 작업 실행

워크플로 디자이너가 작업을 워크플로에 통합하기 전에 작업을 실행하려고 합니다.

사전 요구 사항

작업을 생성합니다. [vRealize Orchestrator 클라이언트에서 작업 생성](#)의 내용을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 라이브러리 > 작업으로 이동한 후 실행할 작업을 선택합니다.
- 3 실행을 클릭합니다.
- 4 필수 입력 매개를 입력하고 실행을 클릭합니다.

작업 실행이 완료되면 **결과/입력** 탭을 클릭합니다. 작업 실행 중 오류가 발생하면 오류가 이 탭에 빨간 색으로 표시됩니다. 작업 실행의 세부 정보는 **작업 결과** 요소에서 확인할 수 있습니다.

참고 작업 실행의 결과는 저장되지 않습니다.

vRealize Orchestrator 클라이언트에서 작업 디버깅

워크플로 디자이너는 스크립트에 중단점을 삽입하여 작업을 디버깅할 수 있습니다.

vRealize Orchestrator에는 작업의 스크립트 및 입력 속성을 디버깅하는 데 사용할 수 있는 기본 제공 디버깅 도구가 포함되어 있습니다. 작업 편집기에서 작업의 스크립트 줄에 중단점을 삽입하여 디버그 프로세스를 시작할 수 있습니다.

참고 기본 제공 디버깅 도구는 기본 JavaScript 런타임을 사용하는 작업에서만 작동합니다. 다른 런타임을 사용하는 작업 스크립트를 디버깅하는 방법에 대한 예는 [Amazon Web Services 작업 디버깅](#) 항목을 참조하십시오.

사전 요구 사항

작업을 생성합니다. [vRealize Orchestrator 클라이언트에서 작업 생성](#)의 내용을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 라이브러리 > 작업으로 이동한 후 디버깅할 작업을 선택합니다.
- 3 작업 편집기에서 디버깅하려는 작업 스크립트의 줄에 중단점을 추가합니다.
- 4 디버그를 클릭합니다.
- 5 작업의 입력 매개 변수를 입력하고 실행을 클릭합니다.

디버그 모드에서 작업 실행이 시작됩니다.

6 중단점에 도달한 후 작업 실행이 일시 중지되면 다음 옵션 중 하나를 선택합니다.

옵션	설명
계속	다른 중단점에 도달하거나 작업 실행이 완료될 때까지 작업 실행을 재개합니다.
한 단계씩 코드 실행	현재 작업 기능을 단계별로 실행합니다. 디버거가 함수의 현재 줄로 더 깊게 이동할 수 없는 경우 프로시저 단위 실행 작업을 수행합니다.
프로시저 단위 실행	디버거가 현재 함수의 다음 줄로 계속됩니다.
메서드에서 리턴	현재 함수가 반환될 때 수행되는 줄로 디버거가 이동합니다.

7 (선택 사항) **디버거** 탭에서 표현식을 추가합니다.

8 (선택 사항) **디버거** 탭에서 변수의 값을 편집합니다.

Python, Node.js 및 PowerShell 스크립트에 대한 핵심 개념

vRealize Orchestrator에서 사용할 스크립트를 생성하는 경우 스크립트의 구조와 형식이 올바른지 확인해야 합니다.

지원되는 런타임

vRealize Orchestrator 8.1부터는 다음 런타임을 사용할 수 있습니다.

- Python 3.7
- Node.js 12
- PowerCLI 11/Powershell 6.2
- PowerCLI 12/Powershell 7

참고 PowerCLI 런타임에는 PowerShell을 비롯해 VMware.PowerCLI, PowerNSX 및 PowervRA 모듈이 포함됩니다.

사용자 지정 소스 코드를 새 런타임에 추가할 수 있지만 컨텍스트와 입력을 수락하고 vRealize Orchestrator 엔진과 결과를 주고 받으려면 올바른 함수 형식을 따라야 합니다.

스크립팅 권장 사항

보다 간단한 스크립팅 작업을 위해 **스크립팅 가능한 작업** 요소를 워크플로 스키마에 추가할 수 있습니다. 보다 복잡한 스크립팅 작업의 경우에는 vRealize Orchestrator 작업을 사용할 수 있습니다.

작업을 사용하면 다음과 같은 두 가지 이점이 있습니다.

- 동작은 워크플로와 별개로 생성하고, 업데이트하고, 가져오고, 내보낼 수 있습니다.
- 동작은 사용자 환경에서 실행 및 디버깅될 수 있는 독립형 개체이므로 개발 프로세스를 원활하게 진행하는 데 유용합니다. [작업 실행 및 디버깅](#)의 내용을 참조하십시오.

스크립트 함수 요구 사항

스크립트 함수의 기본 이름은 **처리기**입니다. 이 함수는 컨텍스트와 입력이라는 두 가지 인수를 받아들입니다. 컨텍스트는 시스템 정보를 포함하는 맵 개체입니다. 예를 들어 `vroURL`은 호출하려는 vRealize Orchestrator 인스턴스의 URL을 포함할 수 있고 `executionId`는 워크플로 실행의 토큰 ID를 포함합니다.

입력은 작업에 제공된 모든 입력을 포함하는 맵 개체입니다. 예를 들어 작업에서 `myInput`이라는 입력을 정의하면, 런타임에 따라 `inputs.myInput` 또는 `inputs["myInput"]`과 같은 입력 인수에서 입력에 액세스 할 수 있습니다. 함수에서 반환하는 모든 항목은 작업의 결과입니다. 따라서 작업의 반환 유형은 스크립트가 vRealize Orchestrator에서 반환하는 콘텐츠 유형과 일치해야 합니다. 기본 숫자를 반환하는 경우 작업 반환 유형은 숫자 유형이어야 합니다. 문자열을 반환하는 경우 작업 반환 유형은 문자열 유형이어야 합니다. 복합 개체를 반환하는 경우 반환 유형은 **Properties** 또는 **Composite Type**에 매핑되어야 합니다. 이와 동일한 원칙이 배열에도 적용됩니다.

Python, Node.js 및 PowerShell 런타임에 지원되는 입력 및 출력 매개 변수 유형:

- String
- Number
- Boolean
- Date
- Properties
- Composite Type

항목 처리기 정의

기본적으로 항목 처리기의 값은 `handler.handler`입니다. 이 값은 vRealize Orchestrator 엔진이 ZIP 패키지에서 `handler`라는 함수가 포함된 `handler.py`, `handler.js` 또는 `handler.ps1`이라는 최상위 파일을 찾는다는 의미입니다. 함수와 처리기 파일의 이름에 차이가 있으면 항목 처리기의 값에 반영되어야 합니다. 예를 들어 기본 처리기 이름이 `index.js`이고 함수 이름이 `callMe`이면, 항목 처리기의 값을 `index.callMe`로 설정해야 합니다.

외부 IDE에서 런타임 스크립트 디버깅

vRealize Orchestrator는 외부 IDE에서 Python 및 Node.js 스크립트의 디버깅을 지원합니다. PowerShell 스크립트는 외부 IDE에서 디버깅할 수 없습니다.

Python, Node.js 및 PowerShell 스크립트에 대한 런타임 제한

일부 Python, Node.js 또는 PowerShell 스크립트를 실행하려면 vRealize Orchestrator Client에서 메모리 및 시간 초과 값을 변경해야 할 수도 있습니다.

vRealize Orchestrator Client는 Python, Node.js 및 PowerShell 작업 스크립트에 기본 메모리 및 시간 초과 값 집합을 사용합니다.

- 메모리: 64MB

■ 시간 초과: 180초

작업 스크립트가 이러한 기본값 중 하나 이상을 초과하면 작업 실행이 실패합니다. 예를 들어, 작업 스크립트가 여러 타사 종속성 모듈을 사용하는 경우가 있습니다. 이러한 시나리오에서는 기본 메모리 제한인 64MB가 충분하지 않을 수 있습니다.

리소스가 부족하여 실패한 작업이 실행되는 것을 방지하려면 작업 편집기에서 메모리 및 시간 초과 값을 변경합니다.

참고 스크립트를 워크플로에 추가할 수 있는 스크립팅 가능한 여러 작업 요소로 분할하는 것을 고려할 수 있습니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 **라이브러리 > 작업**으로 이동하고 작업을 선택합니다.
- 3 **스크립트** 탭을 선택합니다.
- 4 **런타임 제한**에서 메모리 및 시간 초과 값을 변경합니다.
- 5 **저장**을 클릭합니다.
- 6 새로운 런타임 제한을 테스트하려면 **디버그**를 클릭합니다.

구성 요소 관리

7

구성 요소는 전체 vRealize Orchestrator 서버 배포에서 상수를 구성하는 데 사용할 수 있는 변수 목록입니다.

구성 요소를 사용하여 vRealize Orchestrator 서버에서 실행되는 모든 워크플로, 작업 및 정책이 변수를 사용할 수 있게 설정할 수 있습니다.

구성 요소의 변수를 사용하는 워크플로, 작업 또는 정책이 포함된 패키지를 만드는 경우 vRealize Orchestrator에서 해당 패키지에 구성 요소를 자동으로 포함합니다. 구성 요소가 포함된 패키지를 다른 vRealize Orchestrator 서버로 가져오는 경우 구성 요소 변수 값도 가져올 수 있습니다. 예를 들어 실행되는 vRealize Orchestrator 서버에 종속된 변수 값이 필요한 워크플로를 만드는 경우 구성 요소에 이러한 변수를 설정하면 다른 vRealize Orchestrator 서버에서 사용할 수 있도록 해당 워크플로를 내보낼 수 있습니다. 따라서 구성 요소를 사용하면 서버 간에 워크플로, 작업 및 정책을 보다 쉽게 교환할 수 있습니다.

참고 vRealize Orchestrator 5.1 이하에서 내보낸 구성 요소에서는 구성 요소 변수 값을 가져올 수 없습니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트에서 구성 요소 생성](#)

vRealize Orchestrator 클라이언트에서 구성 요소 생성

구성 요소를 사용하여 vRealize Orchestrator 서버에서 공통 변수를 설정할 수 있습니다. 서버에서 실행되는 모든 요소는 구성 요소에서 설정한 변수를 사용할 수 있습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **자산 > 구성**으로 이동합니다.
- 3 **새 구성**을 선택합니다.
- 4 구성 요소 이름을 입력합니다.
- 5 **변수** 탭을 선택합니다.

6 로컬 변수를 생성하려면 **새로 만들기**를 클릭합니다.

- a 변수 이름을 입력합니다.
- b 변수 유형을 선택합니다.

참고 구성 변수 어레이를 생성하려면 **어레이** 확인란을 선택합니다.

- c (선택 사항) 구성 변수의 값을 입력합니다.
- d **저장**을 클릭합니다.

7 구성 요소 생성을 완료하려면 **저장**을 클릭합니다.

다음에 수행할 작업

구성 요소를 사용하여 워크플로, 작업 또는 정책에 변수를 제공할 수 있습니다.

정책은 시스템의 활동을 모니터링하는 이벤트 트리거입니다. 정책은 특정 vRealize Orchestrator 개체의 상태나 성능 변화에 따라 발생하는 미리 정의된 이벤트에 응답합니다.

정책은 vRealize Orchestrator나 vRealize Orchestrator가 플러그인을 통해 액세스하는 기술에서 미리 정의된 특정 이벤트가 발생하면 특정 워크플로나 스크립트를 실행하는 일련의 규칙, 게이지, 임계값 및 이벤트 필터입니다. vRealize Orchestrator는 정책이 실행되는 동안 정책 규칙을 지속적으로 평가합니다. 예를 들어 VC:HostSystem 및 VC:VirtualMachine 유형의 vCenter Server 개체의 동작을 모니터링하는 정책 게이지와 임계값을 구현할 수 있습니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트에서 정책 생성 및 적용](#)
- [vRealize Orchestrator 클라이언트의 정책 요소](#)
- [vRealize Orchestrator 클라이언트에서 정책 실행 관리](#)

vRealize Orchestrator 클라이언트에서 정책 생성 및 적용

정책을 사용하여 특정 이벤트에 대한 vRealize Orchestrator 시스템 작업을 모니터링할 수 있습니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 라이브러리 > 정책으로 이동합니다.
- 3 새 정책을 선택합니다.
빈 정책이 생성되었습니다.
- 4 정책 이름과 버전 번호를 입력합니다.
- 5 변수 탭을 선택합니다.

6 로컬 변수를 생성하려면 **새로 만들기**를 클릭합니다.

- a 변수 이름을 입력합니다.
- b 변수 유형을 선택합니다.

참고 정책 변수 어레이를 생성하려면 **어레이** 확인란을 선택합니다.

- c 변수 값을 입력합니다.

참고 구성 요소 변수의 값을 가져오려면 **구성에 바인딩**을 사용하면 됩니다.

- d **저장**을 클릭합니다.

7 **정의** 탭에서 정책 요소를 추가하고 이벤트 처리기를 설정합니다.

정책 요소에 대한 자세한 내용은 [vRealize Orchestrator 클라이언트의 정책 요소](#) 항목을 참조하십시오.

8 **저장**을 클릭합니다.

정책이 구성되었습니다.

다음에 수행할 작업

정책을 시작하려면 정책을 선택하고 **실행**을 클릭합니다. 정책 실행 이름을 입력하고 메시지가 표시되면 필수 입력 매개 변수를 입력합니다.

정책 상태를 보려면 **작업 > 정책 실행**으로 이동합니다.

vRealize Orchestrator 클라이언트의 정책 요소

정책 요소를 사용하여 이벤트가 발생할 때 미리 정의된 vRealize Orchestrator 워크플로 또는 스크립트를 실행할 수 있습니다.

정책 요소를 추가하여 개체에 의해 트리거된 이벤트에 대한 응답으로 워크플로나 스크립트 실행을 트리거할 수 있습니다. 주기적인 이벤트 요소를 사용하여 워크플로 또는 스크립트 실행을 스케줄링할 수 있습니다. 루트 요소를 사용하여 정책의 시작 또는 중지 동작을 설정할 수 있습니다. 정책 요소에는 정책 요소를 실행해야 하는 시기를 정의하는 이벤트 처리기가 있을 수 있습니다.

참고 정책 요소를 활성화하는 이벤트 처리기는 워크플로 또는 작업 스크립트일 수 있습니다. 이벤트 처리기에 워크플로와 스크립트를 모두 추가하면 정책에서 스크립트 트리거를 무시하고 워크플로 트리거만 사용합니다.

이벤트 처리기	설명
OnInit	정책을 시작할 때마다 정책 요소가 트리거됩니다.
OnExit	정책을 중지할 때마다 정책 요소가 트리거됩니다.
OnExecute	주기적인 이벤트 요소에 사용됩니다. 주기적인 이벤트 요소에 지정된 시간 동안 정책 요소를 트리거합니다.

참고 vRealize Orchestrator 데이터베이스에 연결된 기술에는 고유한 이벤트 처리기가 있을 수 있습니다. 예를 들어 SNMP 플러그인을 통해, SNMP 기반 정책 요소를 생성할 때 **OnTrap** 이벤트 처리기를 사용할 수 있습니다.

정책 요소는 정책 편집 창의 **정의** 탭에서 구성됩니다.

vRealize Orchestrator 클라이언트에서 정책 실행 관리

vRealize Orchestrator Client를 사용하여 vRealize Orchestrator 서버가 다시 시작될 때의 정책에 대한 서버 시작 동작과 정책 우선 순위를 관리할 수 있습니다.

사전 요구 사항

정책을 생성하고 실행합니다. 자세한 내용은 [vRealize Orchestrator 클라이언트에서 정책 생성 및 적용](#) 항목을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 2 **작업 > 정책 실행**으로 이동합니다.
- 3 관리할 정책 실행을 클릭합니다.
- 4 **중지**를 클릭합니다.
정책 상태가 **중지됨**으로 변경됩니다.
- 5 **일반** 탭에서 정책 우선 순위 및 서버 시작 동작을 설정합니다.
- 6 정책을 다시 시작하려면 **실행**을 클릭합니다.
정책 상태가 **실행 중**으로 변경됩니다.

리소스 요소 관리

9

워크플로는 vRealize Orchestrator와 별개로 생성한 개체를 특성으로 사용할 수 있습니다. 워크플로에서 외부 개체를 특성으로 사용하려면 해당 개체를 리소스 요소로 서버에 가져옵니다.

vRealize Orchestrator 워크플로에서 리소스 요소로 사용할 수 있는 개체에는 이미지 파일, 스크립트, XML 템플릿, HTML 파일 등이 있습니다. vRealize Orchestrator 서버에서 실행되는 모든 워크플로에서는 vRealize Orchestrator로 가져온 모든 리소스 요소를 사용할 수 있습니다.

개체를 리소스 요소로 vRealize Orchestrator에 가져온 후에 단일 위치에서 개체를 변경하고 이러한 변경 내용을 해당 리소스 요소를 사용하는 모든 워크플로에 자동으로 전파할 수 있습니다.

리소스 요소의 최대 크기는 16MB입니다.

리소스 요소를 가져오고, 내보내고, 복원하고, 업데이트하고, 삭제할 수 있습니다.

vRealize Orchestrator Client를 사용하여 패키지 생성, 내보내기 및 가져오기를 수행할 수 있습니다. 패키지를 사용하여 다른 vRealize Orchestrator 인스턴스에서 사용할 워크플로 개체를 내보낼 수 있습니다.

패키지는 워크플로, 작업, 정책, 구성 요소 또는 리소스 요소를 포함할 수 있습니다.

패키지에 요소를 추가하면 vRealize Orchestrator가 종속성을 검사하고 패키지에 종속 요소를 추가합니다. 예를 들어 작업이나 다른 워크플로를 사용하는 워크플로를 추가하면 vRealize Orchestrator가 해당 작업과 워크플로를 패키지에 추가합니다.

패키지를 가져오면 서버는 그 콘텐츠에서 서로 다른 요소의 버전을 일치하는 로컬 요소와 비교합니다. 비교 결과는 로컬 요소와 가져온 요소 간의 버전 차이를 보여줍니다. 사용자는 패키지를 가져올지 결정하거나 가져올 특정 요소를 선택할 수 있습니다.

리소스 요소를 제외하고, vRealize Orchestrator Client에서 생성된 대부분의 개체의 경우 패키지가 이러한 개체를 내보내고 가져오는 유일한 방법입니다.

패키지는 디지털 권한 관리를 사용하여 받는 서버에서 패키지의 콘텐츠를 사용하는 방법을 제어합니다. vRealize Orchestrator는 패키지에 서명하고 데이터 보호를 위해 패키지를 암호화합니다. 패키지는 X509 인증서를 사용하여 요소를 내보내고 재배포하는 사용자를 추적할 수 있습니다.

vRealize Orchestrator 클라이언트에서 패키지 생성

패키지에서 워크플로, 정책, 작업, 플러그인 참조, 리소스 요소 및 구성 요소를 내보내고 가져올 수 있습니다. 패키지 개체와 관련된 모든 종속 요소가 패키지에 자동으로 추가되므로 버전 간의 호환성이 유지됩니다. 종속 요소를 삭제하려면 먼저 관련 패키지 개체를 제거해야 합니다.

리소스 요소를 제외하고, vRealize Orchestrator Client에서 생성된 대부분의 개체의 경우 패키지가 이러한 개체를 내보내고 가져오는 유일한 방법입니다.

사전 요구 사항

패키지에 추가할 수 있는 워크플로, 작업, 정책과 같은 개체가 vRealize Orchestrator 서버에 포함되어 있는지 확인합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.

- 2 **자산 > 패키지**로 이동합니다.
- 3 **새 패키지**를 클릭합니다.
- 4 **일반** 탭에서 패키지에 대한 이름과 설명을 입력합니다.

참고 vRealize Orchestrator Client에서 패키지 이름을 지정할 때 특수 문자를 사용할 수 없습니다.

- 5 **컨텐츠** 탭에서 **추가**를 클릭합니다.
- 6 패키지에 추가할 개체를 선택하고 **추가**를 클릭합니다.

참고 종속 요소가 패키지에 자동으로 추가되지만 패키지 생성 중 **컨텐츠** 탭에 표시되지 않습니다. 종속 요소를 보려면 패키지 생성 후 **컨텐츠** 탭을 선택합니다.

- 7 패키지 생성을 완료하려면 **생성**을 클릭합니다.

vRealize Orchestrator 클라이언트에서 패키지 내보내기

vRealize Orchestrator Client를 사용하여 패키지를 다른 vRealize Orchestrator 환경으로 내보낼 수 있습니다.

사전 요구 사항

내보낼 vRealize Orchestrator 개체를 포함하는 패키지를 생성합니다. 자세한 내용은 [vRealize Orchestrator 클라이언트에서 패키지 생성](#) 항목을 참조하십시오.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **자산 > 패키지**로 이동합니다.
- 3 패키지에서 **내보내기**를 클릭합니다.
- 4 (선택 사항) 추가 내보내기 옵션을 선택합니다.

옵션	설명
패키지에 구성 특성 값 추가	구성 요소의 특성 값을 내보냅니다.
패키지에 구성 SecureString 특성 값 추가	SecureString 구성 특성 값을 내보냅니다.
패키지에 글로벌 태그 추가	글로벌 태그를 내보냅니다.

5 패키지를 가져오는 사용자에게 대한 액세스 권한을 설정합니다.

옵션	설명
컨텐츠 보기	사용자가 패키지 콘텐츠를 볼 수 있습니다.
패키지에 추가	사용자가 가져온 패키지의 콘텐츠를 다른 패키지에 추가할 수 있습니다.
컨텐츠 편집	사용자가 패키지 콘텐츠를 편집할 수 있습니다.

6 확인을 클릭합니다.

참고 .package 확장자가 있는 파일이 로컬 시스템의 기본 폴더에 저장됩니다. 사용자 지정 폴더를 설정하려면 브라우저에서 스토리지 설정을 변경하면 됩니다.

결과

패키지를 내보냈습니다. 내보낸 개체를 이제 다른 vRealize Orchestrator 환경에서 사용할 수 있습니다.

vRealize Orchestrator 클라이언트에서 패키지 가져오기

vRealize Orchestrator Client를 사용하여 워크플로 패키지를 가져올 수 있습니다. 패키지를 가져오면 한 vRealize Orchestrator 서버의 개체를 다른 서버에서 재사용할 수 있습니다.

사전 요구 사항

- 수정한 표준 vRealize Orchestrator 요소를 백업합니다.
- 원격 서버에서 가져올 개체가 있는 패키지를 생성하고 내보냅니다.

절차

- 1 vRealize Orchestrator 클라이언트에 로그인합니다.
- 2 **자산 > 패키지**로 이동합니다.
- 3 **가져오기**를 클릭하고, 가져오려는 .package 파일을 찾은 다음 **열기**를 클릭합니다.
- 4 가져온 패키지 정보를 검토합니다.
 - a **일반** 탭에는 이름, 설명, 포함된 항목의 수 및 인증서 정보와 같은 가져온 패키지에 대한 정보가 포함되어 있습니다.
파일을 가져오기 전에 먼저 소스 vRealize Orchestrator 인스턴스의 게시자 인증서를 신뢰하는지를 표시하라는 메시지가 나타날 수 있습니다.
 - b **패키지 요소** 탭에는 가져오기 파일에 포함된 개체가 나열됩니다. 패키지 내 개체의 버전이 서버에 있는 버전보다 상위 버전인 경우 해당 개체 버전을 가져오도록 시스템에서 선택됩니다. 이전 버전의 vRealize Orchestrator 요소는 수동으로 선택해야 합니다.
 - c 패키지의 구성 요소 특성 값을 가져오지 않으려는 경우 **구성 특성 값 가져오기**를 선택 해제합니다.
 - d 드롭다운 메뉴에서 태그를 가져올지 여부를 선택합니다.

5 가져오기를 클릭합니다.

vRealize Orchestrator 클라이언트 에서 문제 해결

11

메트릭, 토큰 재생, 검증 및 디버깅을 사용하여 vRealize Orchestrator 인스턴스를 문제 해결하고 모니터링할 수 있습니다.

본 장은 다음 항목을 포함합니다.

- [vRealize Orchestrator 클라이언트의 메트릭 데이터](#)
- [vRealize Orchestrator Client에서 워크플로 토큰 재생 사용](#)
- [vRealize Orchestrator 워크플로 유효성 검사](#)
- [vRealize Orchestrator 클라이언트에서 워크플로 스크립트 디버그](#)
- [스키마 요소별 워크플로 디버깅](#)

vRealize Orchestrator 클라이언트의 메트릭 데이터

vRealize Orchestrator 관리자는 워크플로 프로파일링 및 시스템 대시보드 메트릭을 사용하여 vRealize Orchestrator 시스템 및 워크플로의 문제를 해결할 수 있습니다.

프로파일링 기능은 워크플로 실행에 대한 메트릭 데이터를 수집합니다. 워크플로 프로파일링은 기본적으로 사용하도록 설정되어 있습니다. 자동 프로파일링은 **제어 센터 > 확장 속성 > profiler-8.2.0**에서 사용하지 않도록 설정할 수 있습니다.

vRealize Orchestrator Client의 메트릭 데이터에 대한 다른 소스는 시스템 수준 메트릭을 제공하는 시스템 대시보드입니다. 자세한 내용은 [vRealize Orchestrator 시스템 대시보드 사용](#) 항목을 참조하십시오.

vRealize Orchestrator 클라이언트에서 워크플로 프로파일링

vRealize Orchestrator 환경의 문제를 해결하고 최적화하기 위해 워크플로 실행을 프로파일링할 수 있습니다.

vRealize Orchestrator Client의 프로파일링 기능을 사용하여 워크플로 실행에 대한 유용한 메트릭 데이터를 수집할 수 있으며, 이 데이터를 사용하여 워크플로의 성능을 최적화할 수 있습니다. 기본적으로 워크플로 실행은 자동으로 프로파일링됩니다. vRealize Orchestrator 제어 센터의 **확장 속성** 페이지에서 자동 프로파일링을 사용하지 않도록 설정하고 프로파일러를 수동으로 실행할 수 있습니다. 수동 프로파일링을 실행하려면 라이브러리에서 워크플로를 찾아서 **작업 > 프로파일**을 선택합니다.

사전 요구 사항

워크플로를 실행합니다.

절차

1 vRealize Orchestrator 클라이언트에 로그인합니다.

2 **작업 > 워크플로 실행**으로 이동합니다.

3 워크플로 실행을 선택합니다.

워크플로 실행 스키마에서 개별 워크플로 항목에 대한 데이터를 볼 수 있습니다. 데이터에는 총 실행 시간, 최대 시간 및 항목 실행 수가 포함됩니다. 페이지 오른쪽 위의 드롭다운 메뉴에서 이 정보를 필터링할 수 있습니다.

4 **성능** 탭을 선택합니다.

이 탭에는 워크플로 실행 CPU 시간, 실행 시간, 토큰 크기 및 워크플로 항목 데이터에 대한 메트릭 데이터가 제공됩니다.

참고 워크플로 실행이 일시 중단되는 경우(예: 워크플로가 추가 입력을 대기할 때) CPU 시간 메트릭은 완료 전에 발생한 런타임 스레드만 캡처합니다.

다음에 수행할 작업

프로파일링에서 수집한 데이터를 사용하여 워크플로를 최적화합니다.

vRealize Orchestrator 시스템 대시보드 사용

관리자는 vRealize Orchestrator Client 시스템 대시보드를 사용하여 vRealize Orchestrator 환경의 노드에 대한 유용한 메트릭 데이터를 수집할 수 있습니다.

시스템 대시보드는 vRealize Orchestrator Client 대시보드 페이지의 맨 위에 있는 **시스템** 탭을 클릭하여 액세스할 수 있습니다. 제공되는 데이터에는 다음이 포함됩니다.

- 노드 상태
- 노드 속성
- 클러스터 설정. 시스템 대시보드에서는 클러스터 설정만 볼 수 있습니다. 이러한 설정을 변경하려면 vRealize Orchestrator 제어 센터의 **Orchestrator 클러스터 관리** 페이지로 이동합니다.
- 스레드 정보
- 힙 메모리
- 힙이 아닌 메모리
- 파일 시스템 사용
- 인증 데이터
- Orchestrator 데이터베이스 연결 풀

■ 프로세스 입력 인수

이 데이터는 vRealize Orchestrator 환경의 개별 노드 상태를 모니터링하고 문제를 해결하는 데 사용할 수 있습니다. 개별 노드 간에 이동하려면 시스템 대시보드 위쪽에 있는 노드와 연결된 탭을 클릭합니다.

vRealize Orchestrator Client에서 워크플로 토큰 재생 사용

토큰 재생 기능을 사용하여 워크플로 실행 항목 간의 전환을 볼 수 있습니다.

토큰 재생 기능은 워크플로 항목 간의 각 전환에 대한 컨텍스트 정보를 기록합니다. 각 워크플로 항목에 대해, 워크플로 실행이 시작되고 종료된 시기와 워크플로 항목 실행이 끝날 때 변경된 변수가 토큰 재생에 기록됩니다. 토큰 재생은 각 워크플로 항목에 대해 생성된 스크립트 로그 메시지도 참조합니다.

참고 워크플로 항목 전환에 대한 데이터는 vRealize Orchestrator PostgreSQL 데이터베이스에 저장됩니다. 이 데이터는 워크플로 실행이 삭제될 때 데이터베이스에서 제거됩니다.

사전 요구 사항

- 제어 센터에서 토큰 재생 기능을 사용하도록 설정합니다.
 - a 제어 센터에 **root**로 로그인합니다.
 - b **확장 속성**을 선택합니다.
 - c **tokenreplay-8.2.0**을 클릭합니다.
 - d 토큰 재생 기능을 사용하도록 설정하려면 **사용**을 클릭합니다.
 - e **저장**을 클릭합니다.

참고 vRealize Orchestrator 서버가 확장을 새로 고치는 데 최대 5분이 걸릴 수 있습니다.

- 워크플로를 실행합니다.

참고 기본적으로 토큰 재생은 vRealize Orchestrator 서버에서 모든 워크플로 실행에 대해 자동으로 실행되지 않습니다. 각 워크플로에 대해 토큰 재생을 개별적으로 실행하거나 제어 센터의 **확장 속성** 페이지에서 모든 워크플로에 대해 토큰 재생 확장을 사용하도록 설정할 수 있습니다.

절차

- 1 (선택 사항) vRealize Orchestrator 서버에서 모든 워크플로 실행에 대해 토큰 재생을 사용하도록 설정합니다.

참고 제어 센터에서 해당 기능을 사용하도록 설정하지 않고 개별 토큰 재생을 실행하려면 워크플로 편집기 페이지에서 **재생을 사용하여 실행**을 클릭합니다.

- a 제어 센터에 **root**로 로그인합니다.
- b **확장 속성**을 선택합니다.
- c **tokenreplay-8.2.0**을 클릭합니다.

- d 모든 워크플로에 대해 토큰 재생 기능을 사용하도록 설정하려면 **모든 워크플로 실행에 대한 재생 기록**을 사용하도록 설정했는지 확인합니다.
- e **저장**을 클릭합니다.

참고 vRealize Orchestrator 서버가 확장을 새로 고치는 데 최대 5분이 걸릴 수 있습니다.

- 2 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 3 **작업 > 워크플로 실행**으로 이동합니다.
- 4 워크플로 실행을 선택합니다.
- 5 왼쪽 메뉴에서 워크플로 실행 항목을 선택합니다.

이제 **변수**와 **로그** 탭에 해당 워크플로 항목에 대한 정보가 표시됩니다.

vRealize Orchestrator 워크플로 유효성 검사

vRealize Orchestrator는 워크플로 유효성 검사 도구를 제공합니다. 워크플로 유효성 검사는 워크플로의 오류를 식별하고 데이터가 한 요소에서 다음 요소로 올바르게 이동하는지 확인하는 데 도움이 됩니다.

기본적으로 워크플로를 실행할 때마다 vRealize Orchestrator는 워크플로 유효성 검사를 항상 수행합니다.

워크플로를 검증할 때 유효성 검사 도구는 오류나 주의 목록을 생성합니다. 목록에서 오류를 클릭하면 오류가 포함된 워크플로 요소가 강조 표시됩니다.

워크플로 편집기에서 유효성 검사 도구를 실행하면 도구가 감지한 오류에 대해 제안하는 빠른 수정 항목을 제공합니다. 일부 빠른 수정에는 추가 정보나 입력 매개 변수가 필요합니다. 오류를 해결해주는 빠른 수정도 있습니다.

워크플로 유효성 검사는 요소 간 데이터 바인딩과 연결을 확인합니다. 워크플로 유효성 검사는 워크플로의 각 요소가 수행하는 데이터 처리를 확인하지는 않습니다. 따라서 스키마 요소의 함수가 잘못된 경우 유효한 워크플로가 잘못 실행되어 잘못된 결과가 생성될 수 있습니다.

vRealize Orchestrator 클라이언트에서 워크플로 검증 및 검증 오류 수정

워크플로를 실행하기 전에 검증해야 합니다. 편집할 워크플로를 연 경우에만 검증 오류를 수정할 수 있습니다.

사전 요구 사항

스키마 요소가 연결되고 바인딩이 정의된 검증하기에 완전한 워크플로인지 확인합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 2 **라이브러리 > 워크플로**로 이동한 후 검증할 워크플로를 선택합니다.
- 3 **편집**을 클릭합니다.

4 맨 위 메뉴에서 **검증**을 클릭합니다.

워크플로가 유효한 경우 확인 메시지가 나타납니다. 워크플로가 잘못된 경우 오류 목록이 나타납니다.

5 잘못된 워크플로의 경우 오류 메시지를 클릭하고 적절한 단계를 수행하여 문제를 해결합니다.

검증 도구는 빨간색 아이콘을 오류가 발생한 스키마 요소에 추가해 강조 표시합니다. 가능한 경우 검증 도구가 빠른 수정 작업을 표시합니다.

- 제안된 빠른 수정 작업에 동의하면 클릭해 해당 작업을 수행합니다.
- 제안된 빠른 수정 작업에 동의하지 않으면 워크플로 검증 대화상자를 닫고 직접 스키마 요소를 수정합니다.

중요 vRealize Orchestrator가 제안하는 수정이 적절한지 항상 확인하십시오.

예를 들어 제안된 작업이 사용하지 않는 특성을 삭제하는 것일 수 있는데, 이때 사실상 해당 특성이 올바르게 바인딩되지 않은 것일 수 있습니다.

6 모든 검증 오류를 제거할 때까지 이전 단계를 반복합니다.

결과

워크플로를 검증하고 검증 오류를 수정했습니다.

다음에 수행할 작업

워크플로를 실행할 수 있습니다.

vRealize Orchestrator 클라이언트에서 워크플로 스크립트 디버그

워크플로 항목의 스크립트에 중단점을 삽입하여 워크플로 실행을 디버깅할 수 있습니다.

중단점에 도달하면 디버깅 프로세스를 계속할 수 있는 몇 가지 옵션이 제공됩니다. 워크플로 스키마에서 요소를 디버깅하면 워크플로 실행에 대한 일반적인 정보를 보고, 워크플로 변수를 수정하고, 감시할 표현식을 추가하고, 로그 메시지를 확인할 수 있습니다.

참고 비운영 환경에서 모든 스크립트 디버깅을 수행합니다.

절차

- 1 vRealize Orchestrator 클라이언트에 관리자로 로그인합니다.
- 2 라이브러리에서 워크플로를 선택합니다.
- 3 워크플로 스키마를 열고 워크플로 요소를 선택한 다음, **스크립팅** 탭을 클릭합니다.
- 4 중단점을 삽입하려면 줄 번호 왼쪽에 있는 빨간색 원을 클릭합니다.

참고 스크립트를 사용하여 워크플로 요소에만 중단점을 삽입할 수 있습니다.

- 5 디버깅 모드에서 워크플로를 실행하려면 **디버그**를 클릭합니다.
워크플로에 입력 매개 변수가 필요하면 제공해야 합니다.
- 6 중단점에 도달한 후 워크플로 실행이 일시 중지되면 사용 가능한 옵션 중 하나를 선택합니다.

옵션	설명
계속	다른 중단점에 도달하거나 워크플로 실행이 완료될 때까지 워크플로 실행을 재개합니다.
한 단계씩 코드 실행	이 옵션을 사용하여 워크플로 요소를 한 단계씩 수행할 수 있습니다. 워크플로 편집기에서 워크플로를 디버깅하는 경우 중첩된 워크플로 요소 내부는 실행할 수 없습니다.
프로시저 단위 실행	스키마의 현재 요소를 건너뛰고 다음 요소에서 워크플로 실행을 일시 중지합니다.

참고 중단점을 클릭하여 현재 중단점을 무시하도록 디버거에 지시할 수 있습니다. 이렇게 하면 중단점 기호가 녹색 삼각형으로 변경됩니다.

- 7 (선택 사항) **디버거** 탭에서 감시할 표현식을 삽입합니다.
표현식을 사용하여 특정 변수의 완료를 추적할 수 있습니다.
- 8 (선택 사항) **디버거** 탭에서 변수의 값을 수정합니다.

스키마 요소별 워크플로 디버깅

워크플로 디자이너는 개별 스키마 요소를 디버깅할 수 있습니다.

절차

- 1 vRealize Orchestrator Client에 로그인합니다.
- 2 **라이브러리 > 워크플로**로 이동하여 워크플로를 선택합니다.
- 3 **스키마** 탭을 선택합니다.
- 4 디버깅할 워크플로 요소를 선택하고 요소의 왼쪽 상단에서 [디버그] 버튼을 클릭합니다.

참고 워크플로 요소 스키마 요소에 중단점을 추가하여 상위 워크플로에서 직접 하위 워크플로를 디버깅할 수 있습니다. 디버거가 **워크플로 요소** 스키마 요소에 도달하면 하위 워크플로의 스키마 보기가 열립니다.

- 5 디버깅하려는 다른 모든 스키마 요소에 대해 반복합니다.
- 6 **디버그**를 클릭합니다.
- 7 요청된 입력 매개 변수 값을 입력하고 **실행**을 클릭합니다.
워크플로 실행이 시작됩니다. 디버거가 중단점이 있는 스키마 요소에 도달하면 워크플로 실행이 일시 중단됩니다.

8 중단점에서 다음 옵션 중 하나를 선택합니다.

옵션	설명
계속	다른 중단점에 도달하거나 워크플로 실행이 완료될 때까지 워크플로 실행을 재개합니다.
한 단계씩 코드 실행	현재 워크플로 기능을 한 단계씩 실행합니다. 디버거가 함수의 현재 줄로 더 깊게 이동할 수 없는 경우 프로시저 단위 실행 작업을 수행합니다.
프로시저 단위 실행	디버거가 현재 함수의 다음 줄로 계속됩니다.
메서드에서 리턴	현재 함수가 반환될 때 수행되는 줄로 디버거가 이동합니다.

9 (선택 사항) 변수 탭에서 워크플로 변수의 값을 편집합니다.

The screenshot shows the VMware vRealize Orchestrator debugger interface. The top panel displays a workflow diagram with a task named "My Orchestrator Task". The right panel shows the "Scripting" tab with the following JavaScript code:

```

1 var len=VM.length;
2 for (var i=0; i < len; i++) {
3   {
4     var VM = VM[i];
5     //var workflowLaunch = Server.getWorkflowId("workflowId");
6     var workflowLaunch = workflowId;
7     if (workflowLaunch == null) {
8       throw "Workflow not found";
9     }
10    var workflowParameters = new Properties();
11    workflowParameters.put("vm", VM);
12    var wToken = workflowLaunch.execute(workflowParameters);
13    System.log ("Run workflow on " + VM.name);

```

The bottom panel shows the "Variables" tab with a table of variables and their values:

Variable	Value
i ①	Not set
VM ②	test
workflowToLaunch ③	Not set
workflowParameters ④	Not set
wToken ⑤	Not set
VM ⑥	test
workflowToRun ⑦	test