

# VMware vRealize Orchestrator용 웹 서비스 클라이언트 개발

vRealize Orchestrator 7.6



vmware®

다음 VMware 웹 사이트에서 최신 기술 문서를 확인할 수 있습니다.

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

VMware 웹 사이트에서는 최신 제품 업데이트도 제공합니다.

본 문서에 대한 의견이 있으시면 다음 주소로 피드백을 보내주십시오.

[docfeedback@vmware.com](mailto:docfeedback@vmware.com)

**VMware, Inc.**

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

# 목차

## VMware vRealize Orchestrator 용 웹 서비스 클라이언트 개발 5

### 1 웹 서비스 클라이언트 개발 6

### 2 vRealize Orchestrator REST API 사용 7

Orchestrator 및 타사 시스템에 대한 인증 8

Orchestrator REST API에서 vCenter Single Sign-On 인증 사용 8

Orchestrator REST API에 대한 참조 설명서 액세스 13

Java REST SDK 사용 13

워크플로를 사용한 작업 14

워크플로 찾기 및 해당 정의 검색 14

워크플로 실행 17

워크플로 프레젠테이션에 대해 입력 매개 변수의 유효성을 검사한 후 워크플로 실행 19

실행 중인 워크플로와의 상호 작용 22

워크플로의 상호 작용 검색 29

워크플로 스키마 액세스 29

작업 사용 29

작업 생성 29

작업 수정 31

작업 상태 확인 31

Orchestrator 인벤토리에서 개체 찾기 32

유형 및 ID로 개체 찾기 32

관계로 개체 찾기 34

필터 적용 34

Orchestrator 개체 가져오기 및 내보내기 35

워크플로 가져오기 35

워크플로 내보내기 36

작업 가져오기 36

작업 내보내기 37

패키지 가져오기 37

패키지 내보내기 38

리소스 가져오기 39

리소스 내보내기 39

구성 요소 가져오기 40

구성 요소 내보내기 40

Orchestrator 개체 삭제 40

워크플로 삭제 41

작업 삭제 41

패키지 삭제	41
리소스 삭제	42
구성 요소 삭제	42
Orchestrator 개체에 대한 사용 권한 설정	43
REST API 사용 권한	43
워크플로 사용 권한 검색	43
워크플로 사용 권한 삭제	44
워크플로에 대한 사용 권한 설정	44
작업 사용 권한 검색	45
작업 사용 권한 삭제	45
작업에 대한 사용 권한 설정	45
패키지 사용 권한 검색	46
패키지 사용 권한 삭제	46
패키지에 대한 사용 권한 설정	47
리소스 사용 권한 검색	47
리소스 사용 권한 삭제	48
리소스에 대한 사용 권한 설정	48
구성 요소 사용 권한 검색	48
구성 요소 사용 권한 삭제	49
구성 요소에 대한 사용 권한 설정	49
플러그인을 사용하여 작업 수행	50
플러그인에 대한 정보 검색	50
플러그인 가져오기	50
플러그인 내보내기	51
플러그인을 사용하거나 사용하지 않도록 설정	51
서버 구성 작업 수행	51
Orchestrator 서버 구성에 대한 정보 검색	51
Orchestrator 서버 구성 가져오기	52
Orchestrator 서버 구성 내보내기	52
태그 지정 작업 수행	53
개체 태그 지정	53
개체 태그 해제	54
개체 태그 나열	54
유형별 태그 지정 개체 나열	54
태그 소유자 나열	55
사용자별 태그 나열	55
태그 이름별로 필터링된 사용자별 태그 나열	55
사용자별 태그 제거	56

# VMware vRealize Orchestrator 용 웹 서비스 클라이언트 개발

VMware vRealize Orchestrator용 웹 서비스 클라이언트 개발에서는 VMware® vRealize Orchestrator용 웹 서비스 클라이언트를 개발하는 방법에 대한 정보를 제공합니다.

Orchestrator는 웹을 통해 워크플로를 액세스하고 사용하는 애플리케이션을 개발할 수 있도록 웹 서비스 API를 제공합니다. Orchestrator는 워크플로를 통해 여러 작업을 수행하는 데 사용할 수 있는 REST(Representational State Transfer) API를 제공합니다.

## 대상 사용자

이 정보는 RESTful 웹 서비스를 통해 네트워크에서 Orchestrator 프로세스에 액세스하려는 웹 애플리케이션 개발자용으로 작성되었습니다.

## 웹 서비스 클라이언트 개발

VMware vRealize Orchestrator는 웹을 통해 워크플로에 액세스하는 애플리케이션을 개발할 수 있도록 웹 서비스 API를 제공합니다. Orchestrator 웹 서비스 API의 주 용도는 사용자 지정 웹 기반 애플리케이션에 Orchestrator 워크플로를 통합할 수 있도록 지원하는 것입니다.

Orchestrator는 REST(Representational State Transfer) API를 기반으로 하는 웹 서비스 API를 제공합니다. Orchestrator REST API는 Orchestrator 인벤토리의 개체 및 설치된 플러그인의 인벤토리를 미리 정의된 URL에서 액세스할 수 있는 리소스로 노출합니다. 이러한 URL에서 HTTP를 요청하면 워크플로를 통해 작업이 트리거됩니다. Orchestrator REST API는 RESTful 웹 서비스 집합을 통해 워크플로 정의를 검색하고, 워크플로를 실행하고, 실행 중인 워크플로의 상태를 확인하고, 워크플로 실행을 취소하고, 대기 중인 사용자 상호 작용을 처리하고, 워크플로 프레젠테이션을 검색하는 등의 작업을 수행하는 데 사용할 수 있는 리소스로 인벤토리 개체를 노출합니다.

# vRealize Orchestrator REST API 사용

## 2

Orchestrator REST API는 HTTP를 통해 Orchestrator 서버와 직접 통신하고 워크플로를 통해 여러 워크플로 관련 작업을 수행할 수 있는 기능을 제공합니다.

Orchestrator REST API는 Orchestrator 서버 및 설치된 플러그인의 인벤토리에 있는 개체를 미리 정의된 URL에 리소스로 노출합니다. 이러한 URL에서 HTTP 호출을 만들어 Orchestrator에서 작업을 트리거할 수 있습니다. 이런 방식으로 워크플로를 통해 여러 작업을 수행할 수 있습니다.

- 워크플로를 실행하고, 워크플로를 스케줄링하고, 워크플로 실행을 검색하고, 사용자 상호 작용에 응답하고, 워크플로 실행을 취소할 수 있습니다.
- 입력 및 출력 매개 변수, 프레젠테이션 등 워크플로에 대한 세부 정보를 검색할 수 있습니다.
- 상태, 생성된 로그, 시작 날짜, 종료 날짜 등 워크플로 실행에 대한 세부 정보를 검색할 수 있습니다.
- Orchestrator 및 설치된 플러그인의 인벤토리를 찾아볼 수 있습니다.
- 워크플로, 작업 및 패키지를 가져오고 내보낼 수 있습니다.

---

**참고** 플러그인 설치에 vRealize Orchestrator REST API를 사용하지 마십시오.

---

Orchestrator REST API를 사용하면 프로그래밍 언어로 빌드할 수 있는 사용자 지정 애플리케이션에 Orchestrator 워크플로를 쉽게 통합할 수 있습니다.

또한 Orchestrator REST API는 응답 데이터 캐싱 메커니즘 및 eTag를 지원합니다.

본 장은 다음 항목을 포함합니다.

- [Orchestrator 및 타사 시스템에 대한 인증](#)
- [Orchestrator REST API에 대한 참조 설명서 액세스](#)
- [Java REST SDK 사용](#)
- [워크플로를 사용한 작업](#)
- [작업 사용](#)
- [Orchestrator 인벤토리에서 개체 찾기](#)
- [Orchestrator 개체 가져오기 및 내보내기](#)
- [Orchestrator 개체 삭제](#)

- Orchestrator 개체에 대한 사용 권한 설정
- 플러그인을 사용하여 작업 수행
- 서버 구성 작업 수행
- 태그 지정 작업 수행

## Orchestrator 및 타사 시스템에 대한 인증

Orchestrator REST API를 통해 만든 HTTP 요청에서 Orchestrator에 대해 인증해야 합니다. Orchestrator REST API를 사용하여 vCenter Server 또는 vRealize Automation과 같은 타사 시스템의 리소스에 액세스하는 경우 해당 시스템에 대해서도 인증해야 합니다.

예를 들어 Orchestrator 인벤토리의 모든 워크플로에 액세스하려면 Orchestrator에 대해 인증해야 합니다. 그러나 vCenter Server에 대해 워크플로를 실행하려면 Orchestrator 및 vCenter Server에 대해 인증해야 합니다.

vRealize Automation 또는 vSphere를 인증 제공자로 포함하여 Orchestrator를 구성하는지 여부에 따라 Orchestrator REST API에 대한 인증 체계가 달라집니다. Orchestrator에서 vCenter Single Sign-On을 사용하는 경우 구성에 따라 vCenter Single Sign-On 서버에서 발급된 holder-of-key 토큰을 사용하여 인증할 수 있습니다. Orchestrator가 vRealize Automation과 함께 구성된 경우 OAuth 전달자 액세스 토큰을 통해 인증할 수 있습니다.

Orchestrator REST API의 최상위 URL에서 HTTP 요청을 만드는 경우에는 Orchestrator에 대해 인증할 필요가 없습니다. Orchestrator REST API의 최상위 URL은 `https://orchestrator_host:port/vco/api/`입니다.

---

**참고** 외부 Orchestrator의 기본 포트 번호는 8281입니다. vRealize Automation에 내장된 Orchestrator 인스턴스에 대한 기본 포트 번호는 443입니다.

---

REST API의 최상위 URL에서 만든 GET 요청은 API를 통해 액세스할 수 있는 모든 리소스에 대한 URL을 반환합니다. 이러한 URL에서 HTTP 요청을 만들려면 Orchestrator에 대해 인증해야 합니다.

## Orchestrator REST API에서 vCenter Single Sign-On 인증 사용

Orchestrator가 vSphere 인증 모드를 사용하여 vCenter Single Sign-On Server와 함께 구성되는 경우 Orchestrator REST API를 통해 Orchestrator의 시스템 개체에 액세스하려면 주체 holder-of-key 토큰이 필요합니다. Orchestrator 서버를 통해 vCenter Single Sign-On Server를 사용하는 vCenter Server 또는 타사 시스템에 액세스하려면 Orchestrator용 대리자 holder-of-key 토큰과 사용자의 주체 토큰이 필요합니다.

Orchestrator가 vCenter Single Sign-On Server와 함께 구성되는 경우 유효한 자격 증명을 사용하여 인증해야 하며, Orchestrator에서 holder-of-key 토큰을 관리합니다.



## Orchestrator의 시스템 개체 액세스

REST API Inventory 및 Catalog Service의 URL에서 Orchestrator의 시스템 개체에 액세스할 수 있습니다.

- `https://orchestrator_host:port/vco/api/inventory/System/`
- `https://orchestrator_host:port/vco/api/catalog/System/`

Orchestrator의 시스템 개체에 액세스하는 경우 HTTP 요청의 Authorization 헤더에서 Inventory 또는 Catalog Service로 주체 holder-of-key 토큰을 전달합니다.

예를 들어 Workflow 유형의 모든 시스템 개체를 검색하려면

`https://orchestrator_host:port/vco/api/catalog/System/Workflow/`에서 GET 요청을 만듭니다. Orchestrator에 대해 인증하려면 요청의 Authorization 헤더에서 주체 holder-of-key 토큰을 전달해야 합니다.

## 타사 시스템의 개체 액세스

Orchestrator REST API를 통해 vCenter Single Sign-On Server에 등록된 타사 시스템에서 작업을 수행하려면 Orchestrator 및 타사 시스템에 대해 인증해야 합니다. Orchestrator REST API를 통해 만든 HTTP 호출에 두 개의 헤더를 포함합니다.

- Authorization. 이 헤더에서 주체 holder-of-key token 토큰을 전달해야 합니다.
- VCOAuthorization. 이 헤더에서 Orchestrator용 대리자 holder-of-key 토큰을 전달해야 합니다. vCenter Single Sign-On Server에서 Orchestrator에 대한 대리자 토큰을 가져와야 합니다. Orchestrator에서는 이 대리자 토큰을 사용하여 사용자를 대신해 타사 시스템에 인증합니다.

예를 들어 Orchestrator REST API를 통해 가상 시스템을 사용하는 워크플로를 실행하려면 Orchestrator와 vCenter Server 양쪽 모두의 리소스에 액세스합니다. Orchestrator 및 vCenter Server에 대해 인증하려면 요청의 Authorization 헤더에서 주체 holder-of-key 토큰을 전달하고 VCOAuthorization 헤더에서 대리자 토큰을 전달해야 합니다. 그러면 사용자는 주체 토큰으로 Orchestrator에 인증하고, Orchestrator는 사용자를 대신해 대리자 토큰으로 vCenter Server에 인증합니다.

vCenter Single Sign-On Server는 Orchestrator를 솔루션으로 처리하며, 모든 솔루션은 고유한 사용자 이름으로 vCenter Single Sign-On Server에 등록됩니다. Orchestrator의 솔루션 사용자 이름과 주체 holder-of-key 토큰을 vCenter Single Sign-On Server에 전달하여 Orchestrator에 대한 대리자 토큰을 요청합니다. vCenter Single Sign-On Server에서 발급한 토큰이 사용자를 대신해 타사 시스템에 인증할 Orchestrator용 대리자 holder-of-key 토큰입니다.

## 예제:vCenter Single Sign-On 모드에서 세션 가져오기

다음 예제 코드는 vCenter Single Sign-On 모드에서 세션을 가져옵니다.

```
URI uri = URI.create("https://orchestrator-server:8281/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

//provide the address of the vCenter Single Sign-On server
```

```

URI ssoUri = URI.create("https://sso-server:7444/ims/STSService?wsdl");

//set the tokens to be valid for an hour
long lifeTimeSeconds = 60 * 60;

//create a factory for vCenter Single Sign-On tokens
SsoAuthenticator sso = new SsoAuthenticator(URI ssoUri, URI adminUri, VcoSessionFactory vcoSessionFactory, long
lifeTimeSeconds);

//provide vCenter Single Sign-On credentials
SsoAuthentication authentication = sso.createSsoAuthentication("username", "password");

VcoSession session = sessionFactory.newSession(authentication);
//use session here

```

## Orchestrator의 솔루션 사용자 이름 가져오기

vCenter Single Sign-On Server는 Orchestrator를 솔루션으로 처리하며, 모든 솔루션은 고유한 사용자 이름으로 vCenter Single Sign-On Server에 등록됩니다. vCenter Single Sign-On Server에서 Orchestrator에 대한 대리자 holder-of-key 토큰을 요청하려면 Orchestrator의 솔루션 사용자 이름이 필요합니다.

### 사전 요구 사항

vCenter Single Sign-On Server에서 발급한 유효한 주체 holder-of-key 토큰이 있는지 확인합니다.

### 절차

- 1 Orchestrator 솔루션 사용자 이름의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/users/
```

- 2 요청의 Authorization 헤더에 주체 holder-of-key 토큰을 제공합니다.

응답의 <user solution-user="OrchestratorSolutionUserName"/> 요소에 Orchestrator의 솔루션 사용자 이름이 포함됩니다. 다음은 Orchestrator 솔루션 사용자 이름의 예입니다.

```
<user xmlns="http://www.vmware.com/vco" admin-rights="true" solution-user="vCO-15d98795afa5b0d6f47ee3aeab3">
```

### 다음에 수행할 작업

Orchestrator 솔루션 사용자 이름과 주체 holder-of-key 토큰을 사용하여 vCenter Single Sign-On Server에서 대리자 holder-of-key 토큰을 요청합니다.

## 구성된 vRealize Automation 인증으로 vRealize Orchestrator REST API SDK 사용

다중 테넌트 또는 단일 테넌트 환경에서 구성된 vRA 인증으로 REST API SDK를 사용할 수 있습니다.

아래 코드에 필요한 인증 토큰(Oauth2.0)을 가져오려면 [Oauth2.0 인증을 사용한 vRO REST API 권한 부여\(2148518\)](#) KB 문서를 참조하십시오.

## 참고 vRealize Automation 인증 모드로 세션 가져오기

다음 예제 코드는 단일 테넌트 및 다중 테넌트 환경 모두에서 vRealize Automation 인증 모드로 세션을 가져옵니다.

- 다중 테넌시가 사용되도록 설정되지 않은 경우:

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTJmM2UyYmYyYi04ZmRmNzY3N"
+ "GZiZWEiLCJwcm4iOiJhZG1pbmIzdHJhdG9yQFZUEhFUKUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXNlc"
+ "MiLCJhdXRoX3RpbWUiOiE1MDIyMDIxMTAsImZyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+ "vbS9TQUFTL3QvbnNwaGVyZS5sb2NhbC9hdXRoIiwiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52bXdhcmUuY29tL"
+ "1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvd2F1dGh0b2t1bGlzImN0eCI6Ilt7XCJtdGRcljpcInVybjpvYXNpczpuYW1lc"
+ "0YzpqTQU1MOjluMDphYzpjbjBGFzc2VzOjBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcljoxNTAyMjAyMT"
+ "n1dliwic2NwIjojdXNlcilslmlkcCI6IjAiLCJlbWwiOiJhZG1pbmIzdHJhdG9yQHNmLT15LTEwLT15LnNvZi1tYnUuZW5nLnZ"
+
+ "td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlInEx6bURnIiwiaWZG1kIjoilwiid2kljoiIiwiaXhwIjoxNTAyMjAyMT"
+ "E1MDIyMDIxMTAsImZyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+ "G9gEQPtmEH5jYab-IITK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCC-
qgKutZl21R"
+ "m740qBKLhmBBONQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//Use the session here
```

- 다중 테넌시가 사용되도록 설정된 경우:

- 일반 테넌트의 사용자:

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTJmM2UyYmYyYi04ZmRmNzY3N"
+
"GZiZWEiLCJwcm4iOiJhZG1pbmIzdHJhdG9yQFZUEhFUKUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXNlc"
+
+ "MiLCJhdXRoX3RpbWUiOiE1MDIyMDIxMTAsImZyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+
+ "vbS9TQUFTL3QvbnNwaGVyZS5sb2NhbC9hdXRoIiwiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52bXdhcmUuY29tL"
+ "1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvd2F1dGh0b2t1bGlzImN0eCI6Ilt7XCJtdGRcljpcInVybjpvYXNpczpuYW1lc"
+
+ "0YzpqTQU1MOjluMDphYzpjbjBGFzc2VzOjBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcljoxNTAyMjAyMT"
+ "n1dliwic2NwIjojdXNlcilslmlkcCI6IjAiLCJlbWwiOiJhZG1pbmIzdHJhdG9yQHNmLT15LTEwLT15LnNvZi1tYnUuZW5nLnZ"
+
+ "td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlInEx6bURnIiwiaWZG1kIjoilwiid2kljoiIiwiaXhwIjoxNTAyMjAyMT"
+ "E1MDIyMDIxMTAsImZyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+ "G9gEQPtmEH5jYab-IITK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCC-
qgKutZl21R"
```

```

Oj"
+ "E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybI90eXBlljoiVVNFUiJ9."
+ "G9gEQPtmEH5jYab-IITK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-
qgKutZl21R"
+ "m740qBKLhmBBONQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//The operations will be executed in the scope of the tenant authenticated with the token above.

//Use the session below

```

#### ■ 솔루션 사용자:

솔루션 사용자는 자신의 테넌트 범위 및 일반 테넌트 범위에서 작업할 수 있습니다. 수행하는 작업의 범위를 재정의할 수 있습니다.

```

URL uri = URL.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

example

String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJNzg4NWNIYS1hZTFmLTJmZUtyYmYyYi04ZmRmNzY3NiIj
+
"GZiZWEiLCJwcm4iOiJhZG1pbmIzdHJhdG9yQFZTUehFukUuTE9DQWwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXNlcI9pZC16Ij
+
"MiLCJhdXRoX3RpbWUiOiJlMDIyMDIxMTAsImIzcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+
"vbS9TQUFTL3QvdnNwaGVyZS5sb2NhbC9hdXRoIiwiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52bXdhcmUuY29t
L"
+ "1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvd2F1dGh0b2tIbIiSlmN0eCI6Ilt7XCJtdGRcljpcInVybjpvYXNpczpuYW1lczp"
+
"OYzpTQU1MOjluMDphYzpjbjbGFzc2VzO1Bhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXJCjYXRcljoxNTAyMjAyMTUwLWw1aWRcljox
M"
+ "n1dliwic2NwIjoidXNlcilslmIkcCI6IjAiLCJlbWwiOiJhZG1pbmIzdHJhdG9yQHNmLTl5LTUwLWw1aWRcljoxNTAyMjAyMTUwLWw1aWRcljox
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlInEx6bURnliwiZG1kIjoiliwid2lkIjoiliwiZXhwIjoxNTAyMjAyMTUwLWw1aWRcljox
Oj"
+ "E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybI90eXBlljoiVVNFUiJ9."
+ "G9gEQPtmEH5jYab-IITK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-
qgKutZl21R"
+ "m740qBKLhmBBONQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

// By default each tenant works in its tenant scope. However, solution users can override the tenant in
which they perform a given operation:
// Here, users of SDK should provide a value that is meaningful to their context.
String overrideWithTenant = "nonSolutionUserTenant";

```

```
VcoSession session = sessionFactory.newSession(auth, overrideWithTenant);

//Use session below
```

## Orchestrator REST API에 대한 참조 설명서 액세스

Orchestrator REST API에 대한 참조 설명서에는 API의 RESTful 웹 서비스, API에 적용되는 데이터 모델, API에 유효한 응답 코드, 코드 예제 등에 대한 정보가 수록되어 있습니다.

Orchestrator REST API에 대한 참조 설명서는 Orchestrator와 함께 설치됩니다. [https://orchestrator\\_host:port/vco/api/docs/](https://orchestrator_host:port/vco/api/docs/)에서 참조 설명서를 사용할 수 있습니다.

공식 Swagger 사양은 <https://swagger.io/specification/>에서 확인할 수 있습니다.

## Java REST SDK 사용

Java SDK 라이브러리를 사용하여 Java 애플리케이션에서 Orchestrator REST API의 작업을 호출하고 개체로 직접 작업할 수 있습니다.

Orchestrator REST SDK의 모든 RESTful 웹 서비스에는 해당 서비스를 사용하여 실행할 수 있는 작업에 해당하는 메시드가 포함된 래핑 Java 클래스가 있습니다.

Java REST SDK는 Orchestrator와 함께 설치됩니다. Java REST SDK 아티팩트는 다음 위치에서 제공됩니다.

---

**참고** Orchestrator Appliance를 배포한 경우에만 이러한 아티팩트에 액세스할 수 있습니다.

---

- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client/](https://orchestrator_host:port/vco-repo/com/vmware/o11n/o11n-rest-client/)
- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-examples/](https://orchestrator_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-examples/)
- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-services/](https://orchestrator_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-services/)
- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/](https://orchestrator_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/)

## 예제: 워크플로를 실행하고 완료될 때까지 대기

다음 예제 코드는 워크플로를 실행하고 해당 워크플로가 완료될 때까지 대기합니다.

```
//start a new session to Orchestrator by using specified credentials
VcoSession session = DefaultVcoSessionFactory.newLdapSession(new URI("https://orchestrator-server:8281/vco/api/"),
"username", "password");

//create the services
WorkflowService workflowService = new WorkflowService(session);
ExecutionService executionService = new ExecutionService(session);
```

```
//find a workflow by ID
Workflow workflow = workflowService.getWorkflow("1231235");

//create an ExecutionContext from the user's input
ExecutionContext context = new ExecutionContextBuilder().addParam("name", "Jerry").addParam("age", 18).build();

//run the workflow
WorkflowExecution execution = executionService.execute(workflow, context);

//wait for the workflow to reach the user interaction state, checking every 500 milliseconds
execution = executionService.awaitState(execution, 500, 10, WorkflowExecutionState.CANCELED,
WorkflowExecutionState.FAILED, WorkflowExecutionState.COMPLETED);

String nameParamValue = new ParameterExtractor().fromTheOutputOf(execution).extractString("name");
System.out.println("workflow was executed with 'name' input set to" + nameParamValue);
```

## 워크플로를 사용한 작업

Orchestrator REST API는 워크플로를 사용하여 여러 작업을 수행하는 데 사용할 수 있는 웹 서비스를 제공합니다.

## 워크플로 찾기 및 해당 정의 검색

워크플로를 사용하여 모든 종류의 작업을 수행하려면 Orchestrator 인벤토리에서 워크플로를 찾고 해당 정의를 검색해야 합니다. 정의에는 워크플로 입력 및 출력 매개 변수가 나열되며, 사용 가능한 워크플로 실행, 워크플로 프레젠테이션 및 기타 개체에 대한 링크가 포함되어 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

### 절차

#### 1 워크플로의 인벤토리 항목을 찾습니다.

- 워크플로의 전체 이름 또는 이름의 키워드를 알고 있는 경우 필터를 적용하여 워크플로의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows?conditions=name={workflowFullName}
```

```
GET https://{orchestrator_host}:{port}/vco/api/workflows?conditions=name~{keyword}
```

- 워크플로 인벤토리 항목의 진입점인 URL에서 GET 요청을 만들어 Catalog 또는 Inventory Service를 통해 워크플로를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/Workflow/
```

```
GET https://{orchestrator_host}:{port}/vco/api/inventory/System/Workflows/
```

- 2 해당 URL에서 GET 요청을 만들어 워크플로의 인벤토리 항목을 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/Workflow/{workflowID}/
```

- 3 정의의 URL에서 GET 요청을 만들어 워크플로 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

## 예제: Send Hello 워크플로 검색

Send Hello 워크플로를 찾고 해당 정의를 검색할 수 있습니다.

- 1 Send Hello 워크플로를 찾으려면 필터를 적용하여 워크플로 서비스의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/workflows?conditions=name~Hello
```

이름에 Hello가 포함된 워크플로 목록이 수신됩니다.

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-items xmlns="http://www.vmware.com/vco" total="2">
  <link rel="down"
    href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080E6808080013086668236014a0614d16e1/">
    <attributes>
      <attribute name="id" value="CF80808080808080808080808080808080E6808080013086668236014a0614d16e1"/>
      <attribute name="canExecute" value="true" />
      <attribute name="description" value="" />
      <attribute name="name" value="Interactive Hello World" />
      <attribute name="type" value="Workflow"/>
      <attribute name="canEdit" value="true"/>
    </attributes>
  </link>
  <link rel="down"
    href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080DA808080013086668236014a0614d16e1/">
    <attributes>
      <attribute name="id" value="CF80808080808080808080808080808080DA808080013086668236014a0614d16e1"/>
      <attribute name="canExecute" value="true" />
      <attribute name="description" value="" />
      <attribute name="name" value="Send Hello" />
      <attribute name="type" value="Workflow"/>
      <attribute name="canEdit" value="true"/>
    </attributes>
  </link>
</inventory-items>
```

- 2 Send Hello 워크플로 인벤토리 항목의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080DA808080013086668236014a0614d16e1/
```

Send Hello 워크플로의 인벤토리 항목이 응답 본문에 수신됩니다.

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-item xmlns="http://www.vmware.com/vco"
    href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF8080808080808080808080DA808080013086668236014a0614d16e1/">
    <relations>
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080DA808080013086668236014a0614d16e1/" />
    </relations>
    <attributes>
        <attribute name="id" value="CF8080808080808080808080DA808080013086668236014a0614d16e1"/>
        <attribute name="canExecute" value="true" />
        <attribute name="description" value="" />
        <attribute name="name" value="Send Hello" />
        <attribute name="type" value="Workflow"/>
        <attribute name="canEdit" value="true"/>
    </attributes>
</inventory-item>
```

3 워크플로 정의를 검색하려면 해당 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e1/
```

Send Hello 워크플로의 정의가 응답 본문에 수신됩니다.

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
  href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/">
  <relations>
    <link rel="up"
      href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
    <link rel="add"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/tasks/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/icon/" />
  </relations>
  <input-parameters>
    <parameter name="name" type="string" />

```





워크플로 정의가 요청의 응답 본문에 수신됩니다.

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
  href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/">
  <relations>
    <link rel="up"
      href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
    <link rel="add"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/tasks/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/icon/" />
  </relations>
  <input-parameters>
    <parameter name="name" type="string" />
  </input-parameters>
  <output-parameters>
    <parameter name="message" type="string" />
  </output-parameters>
  <name>Send Hello</name>
  <description></description>
</workflow>
```

2 워크플로의 실행 개체를 유지하는 URL에서 POST 요청을 만듭니다.

POST https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080DA8080013086668236014a0614d16e1/executions/

요청 본문의 execution-context 요소에 입력 매개 변수의 값을 전달합니다.

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

## 워크플로 프레젠테이션에 대해 입력 매개 변수의 유효성을 검사한 후 워크플로 실행

워크플로 프레젠테이션에서는 워크플로의 입력 매개 변수에 전달할 수 있는 값에 대한 제약 조건(예: 미리 정의된 값 목록 또는 특정 값 범위)을 정의할 수 있습니다. 워크플로를 성공적으로 실행하려면 워크플로 프레젠테이션의 정의에 대해 워크플로의 입력 매개 변수로 전달하는 값의 유효성을 검사해야 합니다.

사용자 지정 애플리케이션에 워크플로를 통합하는 경우 워크플로를 실행할 때 해당 입력 매개 변수 값을 입력할 마법사를 만들어야 할 수 있습니다. 워크플로 프레젠테이션 서비스를 사용하면 워크플로의 프레젠테이션을 인스턴스화하여 마법사의 여러 화면에 해당하는 부분에서 입력 매개 변수 값을 전달할 수 있습니다. 워크플로 프레젠테이션에 정의된 제약 조건에 대해 입력 매개 변수에 전달하는 값의 유효성을 검사할 수 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

### 절차

- 1 워크플로 정의가 포함된 URL에서 GET 요청을 만들어 실행할 워크플로의 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

워크플로의 정의가 요청의 응답 본문에 수신됩니다. 워크플로 정의에서 해당 워크플로의 입력 매개 변수, 워크플로 설명 및 기타 정보를 확인할 수 있습니다.

- 2 해당 URL에서 GET 요청을 만들어 워크플로 프레젠테이션의 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/presentation/
```

- 3 요청의 응답 본문에서 입력 매개 변수에 전달할 수 있는 값에 대한 제약 조건을 기준으로 워크플로 프레젠테이션의 정의를 검토합니다.

예를 들어 입력 매개 변수에 선택할 수 있는 미리 정의된 값 목록이 있을 수 있습니다.

- 4 프레젠테이션 인스턴스의 URL에서 POST 요청을 만들어 워크플로 프레젠테이션을 인스턴스화합니다.

```
POST https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/presentation/instances/
```

- 5 요청 본문에 execution-context 요소를 제공하여 프레젠테이션을 인스턴스화합니다.

빈 execution-context를 전달하거나, 일부 입력 매개 변수 값만 있는 execution-context를 전달할 수 있습니다.

- 6** 입력 매개 변수에 부분적으로 값을 전달하려면 프레젠테이션 인스턴스를 유지하는 URL에서 필요한 수만큼 POST 또는 PUT 요청을 만듭니다.

PUT [https://{orchestrator\\_host}:{port}/vco/api/workflows/{workflowID}/presentation/instances/{executionID}/](https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/presentation/instances/{executionID}/)

- 7 만든 POST 또는 PUT 요청의 응답 본문을 검토합니다.

입력 매개 변수에 전달한 값이 유효한 경우 execution 태그에 valid="true" 특성이 있습니다. 프레젠테이션이 유효한 경우 응답의 out-parameters 요소에 나열된 값을 가져와 워크플로를 실행할 때 입력 매개 변수에 값으로 전달할 수 있습니다.

- 8 입력 매개 변수의 값이 유효한 경우 워크플로 실행을 유지하는 URL에서 POST 요청을 만들어 워크플로를 실행합니다.

POST [https://{orchestrator\\_host}:{port}/vco/api/workflows/{workflowID}/executions/](https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/)

- 9** execution-context 요소에서 워크플로의 입력 매개 변수에 유효한 값을 제공합니다.

**예제:입력 매개 변수의 유효성을 검사하여 Send Hello 워크플로 실행**

해당 프레젠테이션 정의에 대해 입력 매개 변수의 유효성을 검사하여 Send Hello 워크플로를 실행할 수 있습니다.

- 1 Send Hello 워크플로 정의를 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e1/
```

워크플로 정의가 요청의 응답 본문에 수신됩니다.

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
  href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/">
  <relations>
    <link rel="up"
      href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
    <link rel="add"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/tasks/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/icon/" />
```

```
</relations>
<input-parameters>
  <parameter name="name" type="string" />
</input-parameters>
<output-parameters>
  <parameter name="message" type="string" />
</output-parameters>
<name>Send Hello</name>
<description></description>
</workflow>
```

- 2 워크플로 프레젠테이션의 정의를 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/
```

- 3 워크플로 프레젠테이션의 실행 인스턴스를 유지하는 URL에서 POST 요청을 만듭니다.

```
POST https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/instance
s/
```

입력 매개 변수 값을 제공하지 않고 프레젠테이션을 인스턴스화할 수 있도록 빈 execution-context를 제공합니다.

```
<execution-context xmlns="http://www.vmware.com/vco"/>
```

응답 본문의 모든 필드에 입력 매개 변수 값이 잘못되었음을 나타내는 오류 메시지가 추가됩니다.

```
.....
<fields>
  <field type="string" hidden="false" id="name">
    <display-name>name</display-name>
    <description>name</description>
    <messages>
      <message severity="ERROR" code="VCO-CNS0002">
        <Summary>
          The minimum number of characters allowed for this field is 3.0
        </Summary>
      </message>
    </messages>
    <constraints>
      <number-range max="15.0" min="3.0" />
    </constraints>
  </field>
</fields>
.....
```

- 4 특정 프레젠테이션 인스턴스를 유지하는 URL에서 POST 요청을 만듭니다.

```
POST https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/presentation/instance
s/8880808080808080808080808080803F8080800132145338690643f66a027ec/
```

요청 본문에서 입력 매개 변수의 값을 제공합니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

요청의 응답 본문에서 입력 매개 변수의 값이 유효한지 확인할 수 있습니다.

```
<execution started-by="vcoadmin" .... valid="true".....>
```

- 5 프레젠테이션이 유효한 경우 워크플로 실행을 유지하는 URL에서 POST 요청을 만들어 워크플로를 실행합니다.

[illegible]

요청 본문에서 워크플로의 입력 매개 변수에 값을 전달합니다. 워크플로 프레젠테이션의 출력 매개 변수로 반환되는 값을 그대로 사용하거나, 워크플로 프레젠테이션에 대한 마지막 POST 요청의 요청 본문을 직접 사용합니다.

## 실행 중인 워크플로와의 상호 작용

Orchestrator REST API를 사용하여 워크플로가 실행 중인 동안 해당 워크플로에서 여러 작업을 수행할 수 있습니다. 실행 중인 워크플로의 상태를 가져오고, 대기 중인 사용자 상호 작용에 응답하고, 워크플로 실행을 취소할 수 있습니다.

## 워크플로 실행 개체 가져오기 및 워크플로 상태 확인

시작 및 종료 날짜, 실행 상태, 입력 매개 변수 값 등 워크플로 실행에 대한 정보를 가져올 수 있습니다. 또한 워크플로 실행에 대해 생성된 로그를 가져올 수 있습니다.

## 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

## 절차

- 1** 워크플로의 URL에서 GET 요청을 만들어 상태를 확인할 워크플로의 정의를 검색합니다.

GET https://{orchestrator\_host}:{port}/vco/api/workflows/{workflowID}/

워크플로의 정의가 요청의 응답 본문에 수신됩니다. 워크플로 정의에는 해당 워크플로의 실행 인스턴스에 대한 링크가 포함됩니다.

- 2 해당 URL에서 GET 요청을 만들어 워크플로의 사용 가능한 실행 인스턴스를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/
```

요청의 응답 본문에 워크플로의 사용 가능한 실행 인스턴스가 나열됩니다. 여기에서 모든 워크플로 실행의 시작 및 종료 날짜와 상태 및 시작자를 볼 수 있습니다.

- 3 (선택 사항) 특정 워크플로 실행에 대한 자세한 정보를 보려면 해당 실행의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/
```

특정 워크플로 실행의 XML 표현이 요청의 응답 본문에 수신됩니다. 이 실행에 대해 전달된 입력 매개 변수 값, 실행을 시작한 사용자, 시작 및 종료 날짜, 실행 상태 등을 확인할 수 있습니다.

- 4 (선택 사항) 워크플로 실행에 대해 생성된 로그를 검색하려면 로그를 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/logs/
```

- 5 (선택 사항) 워크플로 상태에 대한 추가 정보를 검색하려면 워크플로 상태를 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/state/
```

### 예제: Send Hello 워크플로의 실행 가져오기 및 특정 실행의 상태 확인

Send Hello 워크플로를 실행한 경우 사용 가능한 실행 개체를 가져와 해당 세부 정보를 확인할 수 있습니다.

- 1 정의를 유지하는 URL에서 GET 요청을 만들어 Send Hello 워크플로의 정의를 가져옵니다.

```
GET https://localhost:8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/
```

- 2 워크플로의 실행 개체를 유지하는 URL에서 GET 요청을 만들어 사용 가능한 워크플로 실행을 가져옵니다.

```
GET https://localhost:8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/
```

- 3 요청의 응답 본문에서 워크플로 실행을 선택하고 GET 요청을 만들어 검색합니다.

```
GET https://localhost:8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/executions/8880808080808080808080803A8080800132145338690643f66a027ec/
```

지정된 ID의 워크플로 실행에 대한 XML 표현이 응답 본문에 포함됩니다. 여기에서 해당 실행에 대한 세부 정보를 확인할 수 있습니다

```
.....
<input-parameters>
  <parameter name="name" type="string">
    <string>John Smith</string>
  </parameter>
</input-parameters>
<output-parameters>
  <parameter name="message" type="string">
    <string>Hello, John Smith!</string>
  </parameter>
</output-parameters>
<start-date>2012-01-31T14:28:40.223+03:00</start-date>
<end-date>2012-01-31T14:28:40.410+03:00</end-date>
<started-by>vcadmin</started-by>
<name>Send Hello</name>
.....
```

## 대기 중인 사용자 상호 작용에 응답

Orchestrator REST API를 사용하여 워크플로 실행의 대기 중인 사용자 상호 작용에 응답할 수 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

### 절차

- 1 사용 가능한 사용자 상호 작용 개체를 유지하는 URL에서 GET 요청을 만들거나 대기 중인 사용자 상호 작용만 필터링하여 모든 사용자 상호 작용 개체의 목록을 검색합니다.

URL	설명
<b>https://orchestrator_host:port/vco/api/catalog/System/UserInteraction</b>	Orchestrator에서 사용 가능한 사용자 상호 작용 개체를 유지합니다.
<b>https://orchestrator_host:port/vco/api/catalog/System/UserInteraction?status=0</b>	대기 중인 사용자 상호 작용 개체만 필터링합니다.

사용 가능한 사용자 상호 작용 개체 목록이 수신됩니다. 대기 중인 사용자 상호 작용에는 이름이 state이고 값이 waiting인 특성이 있습니다.

- 2 응답할 대기 중인 사용자 상호 작용의 인벤토리 항목을 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/UserInteraction/{userInteractionID}/
```

인벤토리 항목에 사용자 상호 작용 인스턴스에 대한 링크가 포함됩니다. 사용자 상호 작용 인스턴스는 특정 워크플로 실행과 연결됩니다.



- 3** 특정 워크플로 실행에 대한 사용자 상호 작용 인스턴스의 URL에서 POST 요청을 만듭니다.

POST [https://{orchestrator\\_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/](https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/)

- 4 요청 본문의 execution-context 요소에 사용자 상호 작용의 입력 매개 변수에 대한 값을 제공합니다.

사용자 상호 작용에 성공적으로 응답한 경우 REST API에서 204 상태를 반환합니다.

**예제: 대화형 Hello World 워크플로의 사용자 상호 작용에 응답**

대화형 Hello World 샘플 워크플로를 실행하여 해당 사용자 상호 작용에 응답할 수 있습니다.

- 1 Catalog Service의 사용자 상호 작용 개체에 대한 끝점에서 GET 요청을 만들어 워크플로의 대기 중인 사용자 상호 작용을 검색합니다.

GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0

- 2 대화형 Hello World 워크플로에 대한 사용자 상호 작용 인벤토리 개체를 찾아 해당 URL에서 GET 요청을 만듭니다.

GET https://localhost:  
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080805A8080800132145338690643f66a027ec/

- 3 현재 실행 중인 워크플로 실행에 대한 사용자 상호 작용 개체의 URL에서 POST 요청을 만듭니다.

[illegible]

요청 본문에서 입력 매개 변수에 대한 값을 제공합니다.

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

입력 매개 변수의 유효성을 검사한 후 사용자 상호 작용에 응답

사용자 상호 작용 프레젠테이션에서는 워크플로의 입력 매개 변수에 전달할 수 있는 값에 대한 제약 조건을 정의할 수 있습니다. 사용자 상호 작용에 응답할 때 사용자 상호 작용 프레젠테이션에 정의된 제약 조건에 대해 입력 매개 변수에 전달하는 값의 유효성을 검사할 수 있습니다.

## 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

## 절차

- 1 사용 가능한 사용자 상호 작용 개체를 유지하는 URL에서 GET 요청을 만들거나 대기 중인 사용자 상호 작용만 필터링하여 모든 사용자 상호 작용 개체의 목록을 검색합니다.

URL	설명
<b>https://orchestrator_host:port/vco/api/catalog/System/UserInteraction</b>	Orchestrator에서 사용 가능한 사용자 상호 작용 개체를 유지합니다.
<b>https://orchestrator_host:port/vco/api/catalog/System/UserInteraction?status=0</b>	대기 중인 사용자 상호 작용 개체만 필터링합니다.

사용 가능한 사용자 상호 작용 개체 목록이 수신됩니다. 대기 중인 사용자 상호 작용에는 이름이 state이고 값이 waiting인 특성이 있습니다.

- 2 응답할 대기 중인 사용자 상호 작용의 인벤토리 항목을 유지하는 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/UserInteraction/{userInteractionID}/
```

응답 본문에 사용자 상호 작용 인스턴스의 링크가 포함됩니다. 사용자 상호 작용 인스턴스는 특정 워크플로 실행과 연결됩니다.

- 3 사용자 상호 작용 인스턴스의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/
```

응답 본문에 사용자 상호 작용 프레젠테이션의 다운로드가 있습니다.

- 4 사용자 상호 작용 프레젠테이션의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/presentation/
```

응답 본문에 사용자 상호 작용 프레젠테이션의 정의가 수신됩니다.

- 5 프레젠테이션 정의에서 입력 매개 변수에 전달할 수 있는 값에 대한 제약 조건을 확인합니다.
- 6 프레젠테이션 인스턴스가 있는 URL에서 POST 요청을 만들어 사용자 상호 작용 프레젠테이션을 실행합니다.

```
POST https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/presentation/instances/
```

- 7 요청 본문에서 execution-context 요소의 입력 매개 변수에 대한 값을 제공합니다.

응답 본문에 사용자 상호 작용 프레젠테이션의 인스턴스가 수신됩니다. 입력 매개 변수에 전달한 값이 유효한 경우 execution 요소에 valid="true" 특성이 있습니다. output-parameters 요소에서 사용자 상호 작용에 응답하는 데 사용할 수 있는 입력 매개 변수의 유효한 값을 확인할 수 있습니다.

- 8** 사용자 상호 작용 인스턴스가 있는 URL에서 POST 요청을 만들어 사용자 상호 작용에 응답합니다.

POST [https://{orchestrator\\_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/](https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/)

- 9** 요청 본문에서 입력 매개 변수에 대한 값과 함께 execution-context 컨텍스트를 전달합니다.

사용자 상호 작용 프레젠테이션의 URL에서 만든 POST 요청과 동일한 요청 본문을 사용할 수 있습니다.

마지막 요청에 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

예제:입력 매개 변수의 유효성을 검사하여 대화형 Hello World 워크플로의 사용자 상호 작용에 응답

사용자 상호 작용 프레젠테이션에 정의된 제약 조건에 대해 입력 매개 변수 값의 유효성을 검사하여 대화형 Hello World 워크플로의 사용자 상호 작용에 응답할 수 있습니다.

- 1 Catalog Service의 사용자 상호 작용 개체에 대한 끝점에서 GET 요청을 만들어 워크플로의 대기 중인 사용자 상호 작용을 검색합니다.

GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0

- 2 대화형 Hello World 워크플로에 대한 사용자 상호 작용 인벤토리 개체를 찾아 해당 URL에서 GET 요청을 만듭니다.

GET https://localhost:  
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080805A8080800132145338690643f66a027ec/

- ### 3 사용자 상호 작용 인스턴스의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/catalog/System/UserInteraction/8880808080808080808080805A8080800132145338690643f66a027ec/interaction/
```

- #### 4 사용자 상호 작용 프레젠테이션의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:
8281/vco/api/catalog/System/User Interaction/8880808080808080808080805A8080800132145338690643f66a027e
c/interaction/presentation/
```

이 프레젠테이션은 입력 매개 변수를 필수로 정의하며, 전달할 수 있는 문자열 길이에 대한 제약 조건을 포함합니다.

- 5 사용자 상호 작용 프레젠테이션의 인스턴스를 유지하는 URL에서 POST 요청을 만듭니다.

POST https://localhost:  
8281/vco/api/catalog/System/User Interaction/8880808080808080808080805A8080800132145338690643f66a027e  
c/interaction/presentation/instances/

요청 본문에서 입력 매개 변수에 대한 값을 제공합니다.

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

응답 본문의 execution 요소에 입력 매개 변수 값이 사용자 상호 작용 프레젠테이션의 제약 조건에 대해 유효함을 나타내는 valid="true" 특성이 포함됩니다. 유효한 값이 output-parameters 요소에 나열됩니다.

```

.....
<output-parameters>
  <parameter name="name" type="string">
    <string>John Smith</string>
  </parameter>
</output-parameters>
.....

```

- 6 5단계의 POST 요청과 동일한 요청 본문을 전달하여 사용자 상호 작용 인스턴스의 URL에서 POST 요청을 만듭니다.

POST https://localhost:  
8281/vco/api/catalog/System/UserInteraction/8880808080808080808080805A8080800132145338690643f66a027e  
c/interaction/

## 워크플로 실행 취소

Orchestrator REST API를 사용하여 워크플로 실행을 취소할 수 있습니다.

## 절차

- 1 워크플로 정의 URL에서 GET 요청을 만들어 워크플로 정의를 검색합니다.

GET https://{orchestrator\_host}:{port}/vco/api/workflows/{workflowID}/

워크플로 정의에는 해당 워크플로의 사용 가능한 실행 개체에 대한 링크가 포함됩니다.

- 2 워크플로의 사용 가능한 실행 개체를 유지하는 URL에서 GET 요청을 만들어 사용 가능한 워크플로 실행을 가져옵니다.

GET `https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/`

- 3** 사용 가능한 워크플로 실행 목록에서 취소할 워크플로 실행을 선택하고 해당 URL에서 DELETE 요청을 만듭니다.

DELETE https://{orchestrator\_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/state

## 워크플로의 상호 작용 검색

Orchestrator REST API를 사용하여 워크플로에 대한 모든 사용자 상호 작용 목록을 검색할 수 있습니다.

### 절차

- 1 워크플로 정의 URL에서 GET 요청을 만들어 워크플로 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

- 2 워크플로 상호 작용 URL에서 GET 요청을 만들어 워크플로 상호 작용 목록을 가져옵니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/interactions/
```

GET 요청이 성공하면 상태 코드 200 및 워크플로에 사용 가능한 모든 사용자 상호 작용 목록이 수신됩니다.

## 워크플로 스키마 액세스

Orchestrator REST API를 사용하여 워크플로의 스키마 이미지에 액세스할 수 있습니다.

### 절차

- 1 워크플로 정의 URL에서 GET 요청을 만들어 워크플로 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

- 2 워크플로 스키마 URL에서 GET 요청을 만들어 워크플로 스키마 이미지를 가져옵니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/schema/
```

GET 요청이 성공하면 상태 코드 200과 워크플로 스키마를 나타내는 이미지의 이진 데이터가 수신됩니다. 응답 콘텐츠 유형은 올바른 미디어 유형(예: Content-Type:image/png)으로 설정됩니다.

## 작업 사용

Orchestrator REST API의 작업 서비스를 사용하여 Orchestrator에서 작업 관리와 관련된 모든 작업을 수행할 수 있습니다. 예를 들어 워크플로 스케줄링 작업을 만들고, 기존 작업의 속성을 수정하고, 작업을 삭제할 수 있습니다.

Orchestrator에서 지원되는 예약된 작업의 최대 개수는 50개입니다.

## 작업 생성

Orchestrator REST API를 사용하여 워크플로 스케줄링 작업을 만들 수 있습니다.

## 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

## 절차

- 1 워크플로의 URL에서 GET 요청을 만들어 작업을 생성할 워크플로 정의를 검색합니다.

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

워크플로 정의에서 해당 워크플로의 이름과 ID 및 입력 매개 변수를 확인할 수 있습니다.

- 2 워크플로에 대한 새 작업을 만들려면 작업 서비스의 URL에서 POST 요청을 만듭니다.

```
POST https://{orchestrator_host}:{port}/vco/api/tasks/
```

- 3 요청 본문의 task 요소에 새 작업에 대한 매개 변수를 제공합니다.

요청에 성공하면 API가 상태 코드 202와 빈 응답 본문으로 응답합니다.

## 예제:Send Hello 워크플로에 대한 작업 생성

특정 날짜부터 시작해 매 시간 15분마다 실행되도록 Send Hello 워크플로를 스케줄링하는 작업을 만들 수 있습니다.

- 1 Send Hello 워크플로의 URL에서 GET 요청을 만들어 해당 정의를 검색합니다.

```
GET https://localhost:8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/
```

- 2 요청 본문에 새 작업의 매개 변수를 제공하여 작업 서비스의 URL에서 POST 요청을 만듭니다.

```
POST https://localhost:8281/vco/api/tasks/
```

```
<task xmlns="http://www.vmware.com/vco">
  <name>Send Hello Task</name>
  <recurrence-cycle>every-hours</recurrence-cycle>
  <recurrence-start-date>2012-01-31T11:00:00+00:00</recurrence-start-date>
  <recurrence-end-date>2012-02-05T11:00:00+00:00</recurrence-end-date>
  <recurrence-pattern>15:15</recurrence-pattern>
  <input-parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </input-parameters>
  <workflow href="https://localhost:8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/">
```

```
<name>Send Hello</name>
</workflow>
<start-mode>normal</start-mode>
</task>
```

## 작업 수정

Orchestrator REST API를 사용하여 기존 작업의 속성을 변경할 수 있습니다.

작업에 새 스케줄링 속성을 추가하거나 기존 속성 값을 변경할 수만 있습니다. 작업의 스케줄링 속성을 바꾸려면 해당 작업을 삭제하고 새로 만들어야 합니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

### 절차

- 1 수정할 작업의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/tasks/{task ID}/
```

- 2 요청의 응답 본문에서 작업의 속성을 확인합니다.
- 3 작업을 수정하려면 요청 본문의 task-data 요소에 작업의 새 속성을 제공하여 작업의 URL에서 POST 요청을 만듭니다.

POST 요청이 성공하면 API가 상태 코드 200 및 응답 본문에서 업데이트된 작업을 다시 실행합니다.

## 예제: Send Hello 예제 작업 업데이트

작업의 시작 및 종료 날짜를 업데이트할 수 있습니다. [작업 생성](#)에 소개된 예제 작업을 수정할 수 있습니다. 요청 본문에 새 시작 날짜와 종료 날짜를 제공하여 작업의 URL에서 POST 요청을 만들어야 합니다.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<task-data xmlns="http://www.vmware.com/vco">
  <recurrence-start-date>2012-02-01T14:00:00+02:00</recurrence-start-date>
  <recurrence-end-date>2012-02-05T14:00:00+02:00</recurrence-end-date>
</task-data>
```

## 작업 상태 확인

현재 사용 가능한 작업의 상태를 확인하거나 특정 작업의 모든 실행 인스턴스에 대한 상태를 확인할 수 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

## 절차

- 현재 사용 가능한 모든 작업의 상태를 확인하려면 작업 서비스의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/tasks/
```

응답 본문에 Orchestrator에서 현재 사용 가능한 작업의 정의가 포함됩니다. 모든 작업의 상태는 이름이 state인 attribute 요소에서 확인할 수 있습니다. 요소의 값은 각각 finished, pending, running 등일 수 있습니다.

- 특정 작업의 모든 실행에 대한 상태를 확인하려면 작업 실행이 있는 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/tasks/{taskID}/executions/
```

응답 본문에 해당 작업의 사용 가능한 실행 목록이 수신됩니다. 모든 실행의 상태는 해당 작업 실행 개체의 state 요소에서 확인할 수 있습니다.

## Orchestrator 인벤토리에서 개체 찾기

Catalog 또는 Inventory Service를 사용하여 Orchestrator 인벤토리에서 개체를 찾을 수 있습니다. HTTP 요청을 만드는 URL 끝에 필터 매개 변수를 적용하여 개체의 특정 하위 집합에만 액세스할 수 있습니다.

Catalog Service를 사용하여 Orchestrator 인벤토리에서 특정 유형의 개체를 찾거나, 해당 유형과 ID로 특정 개체를 검색할 수 있습니다. 예를 들어 유형이 workflow 또는 action인 모든 개체를 검색하거나, 특정 워크플로 또는 작업을 검색할 수 있습니다.

Inventory Service를 사용하면 상위-하위 관계를 사용하여 Orchestrator 인벤토리를 찾아볼 수 있습니다. 또한 Inventory Service를 사용하면 Orchestrator 인벤토리의 특정 위치에서 사용 가능한 개체에 액세스할 수 있습니다. 예를 들어 Orchestrator 인벤토리에서 해당 위치(예:

Library/vCenter/Datacenter)로 이동하여 데이터 센터 관리에 대한 모든 워크플로를 검색할 수 있습니다.

Orchestrator REST API의 모든 서비스는 HTTP 요청을 만들 때 URL 끝에 추가할 수 있는 필터 매개 변수를 지원합니다. 필터 매개 변수를 사용하면 특정 URL에서의 요청에 대한 응답 본문에 수신되는 결과의 범위를 좁힐 수 있습니다.

## 유형 및 ID로 개체 찾기

REST API의 Catalog Service를 사용하여 Orchestrator에서 유형 및 ID로 개체를 찾을 수 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.



## 절차

- 1 Catalog Service의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/
```

Orchestrator의 인벤토리를 노출하는 플러그인의 카탈로그 진입점 및 Orchestrator의 시스템 개체에 대한 다운로드가 요청의 응답 본문에 포함됩니다.

- `https://{orchestrator_host}:{port}/vco/api/catalog/{plugin_namespace}/`
- `https://{orchestrator_host}:{port}/vco/api/catalog/System/`

- 2 플러그인에서 노출하는 개체 또는 Orchestrator의 시스템 개체에 액세스하려면 플러그인에 대한 카탈로그 진입점의 URL 또는 Orchestrator의 시스템 개체가 있는 URL에서 GET 요청을 만듭니다.

노출된 개체의 유형에 대한 링크가 요청의 응답 본문에 포함됩니다.

- ### 3 액세스하려는 개체 유형의 URL에서 GET 요청을 만듭니다.

GET https://{orchestrator\_host}:{port}/vco/api/catalog/{namespace}/{objectType}/

- #### 4 찾으려는 특정 개체의 URL에서 GET 요청을 만듭니다.

GET https://{orchestrator\_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/

## 예제:Send Hello 워크플로 찾기

Catalog Service를 사용하여 샘플 Send Hello 워크플로를 찾을 수 있습니다.

- 1 Catalog Service의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/catalog/
```

- 2 Orchestrator의 모든 시스템 개체가 있는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/catalog/System/
```

- 3 모든 워크플로가 있는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/catalog/Workflow/
```

- 4 Send Hello 워크플로의 URL에서 GET 요청을 만듭니다.

GET https://localhost:  
8281/vco/api/catalog/Workflow/CF8080808080808080808080808080DA808080013086668236014a0614d16e1/

## 관계로 개체 찾기

Orchestrator REST의 Inventory Service를 사용하여 Orchestrator 및 플러그인 인벤토리를 계층으로 찾아볼 수 있습니다.

### 사전 요구 사항

Orchestrator에서 샘플 워크플로 패키지를 가져왔는지 확인합니다. 이 패키지는 Orchestrator 설명서 페이지에서 다운로드할 수 있는 Orchestrator 샘플 애플리케이션 ZIP 파일에 포함되어 있습니다.

### 절차

- 1 Inventory Service의 URL에서 GET 요청을 만듭니다.

```
GET https://{orchestrator_host}:{port}/vco/api/inventory/
```

설치된 플러그인의 등록된 인벤토리 및 Orchestrator의 System 아래에 있는 시스템 개체에 대한 다운로드가 응답 본문에 포함됩니다.

- 2 액세스하려는 인벤토리의 다운로드에서 GET 요청을 만듭니다.
- 3 찾으려는 개체에 도달할 때까지 인벤토리 항목의 업링크 및 다운로드에서 GET 요청을 만듭니다.

### 예제:Send Hello 워크플로 찾기

Orchestrator 인벤토리에서 Send Hello 워크플로를 찾을 수 있습니다.

- 1 Inventory Service의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/inventory/
```

- 2 Orchestrator의 시스템 개체가 있는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/inventory/System/
```

- 3 Orchestrator의 모든 워크플로가 있는 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/
```

- 4 샘플 워크플로 범주의 URL에서 GET 요청을 만듭니다.

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/Samples/
```

- 5 Hello World 워크플로 범주의 다운로드를 사용하여 Send Hello 워크플로를 찾습니다.

## 필터 적용

Orchestrator REST API의 서비스는 API에 대한 HTTP 요청에서 반환되는 개체의 범위를 좁힐 수 있는 추가 URL 매개 변수를 지원합니다.

REST API를 통해 액세스할 수 있는 리소스의 모든 URL에 대해 서로 다른 쿼리 매개 변수가 지원됩니다. URL에 적용되는 쿼리 매개 변수를 알아보려면 vRealize Orchestrator REST API 참조 설명서를 참조하십시오.

#### 절차

- ◆ 특정 URL의 요청에서 반환되는 결과의 범위를 좁히려면 해당 URL 끝에 필터를 적용합니다.

URL?filter\_1 & filter\_2 & filter\_3 & .... & filter\_N. 모든 필터는 관련 URL에 유효한 쿼리 매개 변수를 포함합니다. 모든 URL의 유효한 쿼리 매개 변수에 대한 자세한 내용은 Orchestrator REST API 참조 설명서를 참조하십시오.

### 예제: 필터 워크플로

해당 이름에 특정 단어(예: datastore)가 포함된 워크플로를 찾으려는 경우 Catalog Service에 대한 요청에 다음 필터를 적용할 수 있습니다.

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?conditions=name~datastore
```

반환되는 워크플로 수를 특정 개수(예: 5개)로 제한하려면 요청에 추가 필터를 적용합니다.

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?conditions=name~datastore&maxResult=5
```

## Orchestrator 개체 가져오기 및 내보내기

Orchestrator REST API는 워크플로, 작업, 패키지, 리소스 및 구성 요소를 가져오고 내보내는 데 사용할 수 있는 웹 서비스를 제공합니다.

### 워크플로 가져오기

Orchestrator REST API를 사용하여 워크플로를 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 워크플로의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

---

**참고** HTML5 기반 vRealize Orchestrator Client에서 생성된 워크플로의 입력 양식 프레젠테이션은 REST API를 사용하여 가져올 수 없습니다. 워크플로 프레젠테이션을 가져오려면 vRealize Orchestrator Client의 패키지 기능을 사용하십시오.

---

#### 사전 요구 사항

워크플로 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

#### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 워크플로의 속성을 정의합니다.

## 2 워크플로 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/workflows/
```

POST 요청이 성공하면 상태 코드 202가 수신됩니다.

## 워크플로 내보내기

Orchestrator REST API를 사용하여 워크플로를 내보내고 파일로 다운로드할 수 있습니다.

**참고** HTML5 기반 vRealize Orchestrator Client에서 생성된 워크플로의 입력 양식 프레젠테이션은 REST API를 사용하여 내보낼 수 없습니다. 워크플로 프레젠테이션을 내보내려면 vRealize Orchestrator Client의 패키지 기능을 사용하십시오.

### 절차

1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.

- 이름: `accept`
- 값: `application/zip`

2 내보내려는 워크플로의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 워크플로 이진 콘텐츠가 기본 파일 이름이 `workflow_name.workflow`인 첨부 파일로 제공됩니다. REST 클라이언트 애플리케이션에서 파일을 저장할 수 있습니다.

## 작업 가져오기

Orchestrator REST API를 사용하여 작업을 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 작업의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

### 사전 요구 사항

작업 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

### 절차

1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 작업의 속성을 정의합니다.

2 작업 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/actions/
```

POST 요청이 성공하면 상태 코드 202가 수신됩니다.

## 작업 내보내기

Orchestrator REST API를 사용하여 작업을 내보내고 파일로 다운로드할 수 있습니다.

### 절차

1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.

- 이름: accept
- 값: application/zip

2 내보내려는 작업의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 작업 이진 콘텐츠가 기본 파일 이름이 *action\_name.action*인 첨부 파일로 제공됩니다. REST 클라이언트 애플리케이션에서 파일을 저장할 수 있습니다.

## 패키지 가져오기

Orchestrator REST API를 사용하여 패키지를 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 패키지의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

중복된 이름의 Orchestrator 패키지를 가져오는 경우 기본적으로 기존 패키지를 덮어쓰지 않습니다. 요청에서 매개 변수를 사용하여 기존 패키지를 덮어쓸지 지정할 수 있습니다.

Orchestrator 패키지는 기본적으로 구성 요소의 특성 값과 함께 가져옵니다. 그러나 요청에서 매개 변수를 사용하여 특성 값 없이 패키지를 가져올 수 있습니다.

Orchestrator 패키지에 포함된 태그는 기본적으로 가져오지만 동일한 태그가 Orchestrator 서버에 이미 있는 경우 기존 태그의 값이 유지됩니다. 요청에서 매개 변수를 사용하여 기존 태그 값을 유지할지 지정할 수 있습니다.

### 사전 요구 사항

패키지 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

### 절차

1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 패키지의 속성을 정의합니다.

2 패키지 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/
```

- 3 (선택 사항) 패키지를 가져오고 이름이 같은 기존 패키지를 덮어쓰려면 POST 요청에서 `overwrite` 매개 변수를 사용합니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?overwrite=true
```

- 4 (선택 사항) 구성 요소의 특성 값 없이 패키지를 가져오려면 POST 요청에서 `importConfigurationAttributeValues` 매개 변수를 사용합니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?importConfigurationAttributeValues=false
```

- 5 (선택 사항) 포함된 태그 없이 패키지를 가져오려면 POST 요청에서 `tagImportMode` 매개 변수를 사용합니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?tagImportMode=DoNotImport
```

- 6 (선택 사항) 포함된 태그와 함께 패키지를 가져오고 기존 태그 값을 덮어쓰려면 POST 요청에서 `tagImportMode` 매개 변수를 사용합니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?tagImportMode=ImportAndOverwriteExistingValue
```

POST 요청이 성공하면 상태 코드 202가 수신됩니다.

## 패키지 내보내기

Orchestrator REST API를 사용하여 패키지를 내보내고 파일로 다운로드할 수 있습니다.

기본적으로 Orchestrator 패키지는 구성 요소의 특성 값 또는 전역 태그와 함께 내보내집니다. 그러나 요청에서 매개 변수를 사용하여 특성 값 또는 전역 태그 없이 패키지를 내보낼 수 있습니다. 또한 다운로드한 패키지 파일의 사용자 지정 이름을 지정할 수 있습니다.

### 절차

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.

- 이름: `accept`
- 값: `application/zip`

- 2 내보내려는 패키지의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/
```

- 3 (선택 사항) 내보낸 패키지의 사용자 지정 이름을 설정하려면 GET 요청에서 `packageName` 매개 변수를 사용합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?packageName={custom_name}
```

- 4 (선택 사항) 구성 요소의 특성 값 없이 패키지를 내보내려면 GET 요청에서 `exportConfigurationAttributeValues` 매개 변수를 사용합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?
exportConfigurationAttributeValues=false
```

- 5 (선택 사항) 전역 태그 없이 패키지를 내보내려면 GET 요청에서 `exportGlobalTags` 매개 변수를 사용합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?exportGlobalTags=false
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 패키지 이진 콘텐츠가 기본 파일 이름이 `package_name.package`인 첨부 파일로 제공됩니다. REST 클라이언트 애플리케이션에서 파일을 저장할 수 있습니다.

## 리소스 가져오기

Orchestrator REST API를 사용하여 리소스를 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 리소스의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

### 사전 요구 사항

리소스 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 리소스의 속성을 정의합니다.
- 2 리소스 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/resources/
```

POST 요청이 성공하면 상태 코드 202가 수신됩니다.

## 리소스 내보내기

Orchestrator REST API를 사용하여 리소스를 내보낼 수 있습니다.

### 절차

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.
  - 이름: `accept`
  - 값: `application/octet-stream`

## 2 내보내려는 리소스의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 리소스의 콘텐츠가 응답 본문에 제공됩니다.

## 구성 요소 가져오기

Orchestrator REST API를 사용하여 구성 요소를 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 구성 요소의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

### 사전 요구 사항

구성 요소 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 구성 요소의 속성을 정의합니다.
- 2 구성 요소 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/configurations/
```

POST 요청이 성공하면 상태 코드 202가 수신됩니다.

## 구성 요소 내보내기

Orchestrator REST API를 사용하여 구성 요소를 내보낼 수 있습니다.

### 절차

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.
  - 이름: accept
  - 값: application/vcoobject+xml
- 2 내보내려는 구성 요소의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 구성 요소 콘텐츠가 응답 본문에 제공됩니다.

## Orchestrator 개체 삭제

Orchestrator REST API는 워크플로, 작업, 패키지, 리소스 및 구성 요소를 삭제하는 데 사용할 수 있는 웹 서비스를 제공합니다.



## 워크플로 삭제

Orchestrator REST API를 사용하여 워크플로를 삭제할 수 있습니다.

### 절차

- 1 GET 요청을 하고 반환된 워크플로 목록에서 워크플로의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

- 2 워크플로의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

DELETE 요청이 성공하면 상태 코드 200과 빈 응답 본문이 수신됩니다.

## 작업 삭제

Orchestrator REST API를 사용하여 작업을 삭제할 수 있습니다.

### 절차

- 1 GET 요청을 하고 반환된 작업 목록에서 작업의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

- 2 작업의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/
```

DELETE 요청이 성공하면 상태 코드 200과 빈 응답 본문이 수신됩니다.

## 패키지 삭제

Orchestrator REST API를 사용하여 패키지를 삭제할 수 있습니다.

패키지를 삭제하는 경우 패키지의 요소는 삭제되지 않습니다. 패키지의 콘텐츠를 삭제하려면 옵션 매개 변수를 제공해야 합니다.

### 절차

- 1 GET 요청을 하고 반환된 패키지 목록에서 패키지의 이름을 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

- 패키지의 URL에서 DELETE 요청을 만들고, 패키지에서 요소를 삭제하려는 경우 요청 끝에 옵션 매개 변수를 제공합니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/packages/{package_name}?option={parameter}
```

매개 변수	설명
<code>deletePackage</code>	패키지만 삭제되고 해당 콘텐츠는 유지됩니다.
<code>deletePackageWithContent</code>	패키지와 모든 콘텐츠가 삭제됩니다. 다른 패키지에서 삭제된 패키지와 요소를 공유하는 경우 공유된 요소는 다른 패키지에서도 삭제됩니다.
<code>deletePackageKeepingShared</code>	패키지와 공유되지 않는 콘텐츠가 삭제됩니다. 다른 패키지와 공유된 요소는 삭제되지 않습니다.

옵션 매개 변수를 제공하지 않은 경우 기본 `deletePackage` 매개 변수가 사용됩니다.

DELETE 요청이 성공하면 상태 코드 200과 빈 응답 본문이 수신됩니다.

## 리소스 삭제

Orchestrator REST API를 사용하여 리소스를 삭제할 수 있습니다.

### 절차

- GET 요청을 하고 반환된 리소스 목록에서 리소스의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

- 리소스의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/
```

DELETE 요청이 성공하면 상태 코드 200과 빈 응답 본문이 수신됩니다.

## 구성 요소 삭제

Orchestrator REST API를 사용하여 구성 요소를 삭제할 수 있습니다.

### 절차

- GET 요청을 하고 반환된 구성 요소 목록에서 구성 요소의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

- 구성 요소의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/
```

DELETE 요청이 성공하면 상태 코드 200과 빈 응답 본문이 수신됩니다.

## Orchestrator 개체에 대한 사용 권한 설정

REST API를 사용하여 Orchestrator 개체에 대한 사용자 지정 사용 권한을 설정할 수 있습니다. 사용 권한을 설정하려면 개체 사용 권한의 URL에서 POST 요청을 만들고 요청 본문에서 사용 권한을 정의해야 합니다.

또한 Orchestrator REST API를 사용하여 개체의 사용 권한에 대한 정보를 검색하거나 기존 사용 권한을 삭제할 수 있습니다.

### REST API 사용 권한

Orchestrator REST API를 사용하여 사용 권한을 설정하는 경우 문자 집합을 사용하여 사용 권한을 정의해야 합니다.

POST 요청의 요청 본문에 있는 <rights> 태그에 문자 시퀀스를 포함하여 요소에 대한 사용 권한을 정의할 수 있습니다.

Orchestrator REST API를 통해 사용 권한을 설정하는 데 사용할 수 있는 문자에는 특정 의미가 있습니다.

**표 2-1. Orchestrator REST API 사용 권한 문자 집합**

문자	설명
r	보기 권한을 제공합니다.
x	실행 권한을 제공합니다.
i	검사 권한을 제공합니다.
c	편집 권한을 제공합니다.
a	관리 권한을 제공합니다.

### 예제: 사용 권한 설정 구문

Orchestrator 요소 사용 권한의 URL에서 POST 요청의 요청 본문에 다음 예제 구문을 사용할 수 있습니다.

```
<permissions xmlns="http://www.vmware.com/vco">
  <permission>
    <principal>cn=vcousers,ou=vco,dc=appliance</principal>
    <rights>ric</rights>
  </permission>
</permissions>
```

요청 본문의 <rights> 태그에 ric 사용 권한을 설정하여 vcousers 사용자 그룹의 구성원이 Orchestrator 요소를 보고, 검사하고, 편집하도록 허용할 수 있습니다.

### 워크플로 사용 권한 검색

Orchestrator REST API를 사용하여 워크플로 사용 권한에 대한 정보를 검색할 수 있습니다.

## 절차

- 1 GET 요청을 하고 반환된 워크플로 목록에서 워크플로의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

- 2 워크플로 사용 권한의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 워크플로 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 워크플로 사용 권한 삭제

Orchestrator REST API를 사용하여 워크플로 사용 권한을 삭제할 수 있습니다. 새 사용 권한을 설정하기 전에 워크플로의 기존 사용 권한을 삭제할 수 있습니다.

## 절차

- 1 GET 요청을 하고 반환된 워크플로 목록에서 워크플로의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

- 2 워크플로 사용 권한의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

DELETE 요청이 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

## 워크플로에 대한 사용 권한 설정

Orchestrator REST API를 사용하여 워크플로에 대한 사용 권한을 설정할 수 있습니다.

## 사전 요구 사항

설정할 수 있는 사용 권한 유형 및 요청 본문에서 사용할 수 있는 구문을 검토합니다. [REST API 사용 권한](#) 항목을 참조하십시오.

## 절차

- 1 GET 요청을 하고 반환된 워크플로 목록에서 워크플로의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

- 2 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용 권한을 설정할 워크플로의 속성을 정의합니다.

- 3 요청 본문에서 설정할 사용 권한을 지정합니다.

#### 4 워크플로 사용 권한의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

POST 요청이 성공하면 상태 코드 201이 수신됩니다. 워크플로 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

### 작업 사용 권한 검색

Orchestrator REST API를 사용하여 작업 사용 권한에 대한 정보를 검색할 수 있습니다.

#### 절차

- 1 GET 요청을 하고 반환된 작업 목록에서 작업의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

- 2 작업 사용 권한의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 작업 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

### 작업 사용 권한 삭제

Orchestrator REST API를 사용하여 작업 사용 권한을 삭제할 수 있습니다. 새 사용 권한을 설정하기 전에 작업의 기존 사용 권한을 삭제할 수 있습니다.

#### 절차

- 1 GET 요청을 하고 반환된 작업 목록에서 작업의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

- 2 작업 사용 권한의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

DELETE 요청이 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

### 작업에 대한 사용 권한 설정

Orchestrator REST API를 사용하여 작업에 대한 사용 권한을 설정할 수 있습니다.

#### 사전 요구 사항

설정할 수 있는 사용 권한 유형 및 요청 본문에서 사용할 수 있는 구문을 검토합니다. [REST API 사용 권한](#) 항목을 참조하십시오.

**절차**

- 1 GET 요청을 하고 반환된 작업 목록에서 작업의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

- 2 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용 권한을 설정할 작업의 속성을 정의합니다.
- 3 요청 본문에서 설정할 사용 권한을 지정합니다.
- 4 작업 사용 권한의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

POST 요청이 성공하면 상태 코드 201이 수신됩니다. 작업 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

**패키지 사용 권한 검색**

Orchestrator REST API를 사용하여 패키지 사용 권한에 대한 정보를 검색할 수 있습니다.

**절차**

- 1 GET 요청을 하고 반환된 패키지 목록에서 패키지의 이름을 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

- 2 패키지 사용 권한의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 패키지 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

**패키지 사용 권한 삭제**

Orchestrator REST API를 사용하여 패키지 사용 권한을 삭제할 수 있습니다. 새 사용 권한을 설정하기 전에 패키지의 기존 사용 권한을 삭제할 수 있습니다.

**절차**

- 1 GET 요청을 하고 반환된 패키지 목록에서 패키지의 이름을 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

- 2 패키지 사용 권한의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

DELETE 요청이 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

## 패키지에 대한 사용 권한 설정

Orchestrator REST API를 사용하여 패키지에 대한 사용 권한을 설정할 수 있습니다.

### 사전 요구 사항

설정할 수 있는 사용 권한 유형 및 요청 본문에서 사용할 수 있는 구문을 검토합니다. [REST API 사용 권한](#) 항목을 참조하십시오.

### 절차

- 1 GET 요청을 하고 반환된 패키지 목록에서 패키지의 이름을 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

- 2 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용 권한을 설정할 패키지의 속성을 정의합니다.
- 3 요청 본문에서 설정할 사용 권한을 지정합니다.
- 4 패키지 사용 권한의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

POST 요청이 성공하면 상태 코드 201이 수신됩니다. 패키지 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 리소스 사용 권한 검색

Orchestrator REST API를 사용하여 리소스 사용 권한에 대한 정보를 검색할 수 있습니다.

### 절차

- 1 GET 요청을 하고 반환된 리소스 목록에서 리소스의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

- 2 리소스 사용 권한의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 리소스 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 리소스 사용 권한 삭제

Orchestrator REST API를 사용하여 리소스 사용 권한을 삭제할 수 있습니다. 새 사용 권한을 설정하기 전에 리소스의 기존 사용 권한을 삭제할 수 있습니다.

### 절차

- 1 GET 요청을 하고 반환된 리소스 목록에서 리소스의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

- 2 리소스 사용 권한의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

DELETE 요청이 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

## 리소스에 대한 사용 권한 설정

Orchestrator REST API를 사용하여 리소스에 대한 사용 권한을 설정할 수 있습니다.

### 사전 요구 사항

설정할 수 있는 사용 권한 유형 및 요청 본문에서 사용할 수 있는 구문을 검토합니다. [REST API 사용 권한](#) 항목을 참조하십시오.

### 절차

- 1 GET 요청을 하고 반환된 리소스 목록에서 리소스의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

- 2 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용 권한을 설정할 리소스의 속성을 정의합니다.
- 3 요청 본문에서 설정할 사용 권한을 지정합니다.
- 4 리소스 사용 권한의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

POST 요청이 성공하면 상태 코드 201이 수신됩니다. 리소스 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 구성 요소 사용 권한 검색

Orchestrator REST API를 사용하여 구성 요소 사용 권한에 대한 정보를 검색할 수 있습니다.



## 절차

- 1 GET 요청을 하고 반환된 구성 요소 목록에서 구성 요소의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

- 2 구성 요소 사용 권한의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 구성 요소 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 구성 요소 사용 권한 삭제

Orchestrator REST API를 사용하여 구성 요소 사용 권한을 삭제할 수 있습니다. 새 사용 권한을 설정하기 전에 구성 요소의 기존 사용 권한을 삭제할 수 있습니다.

## 절차

- 1 GET 요청을 하고 반환된 구성 요소 목록에서 구성 요소의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

- 2 구성 요소 사용 권한의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

DELETE 요청이 성공하면 상태 코드 204와 빈 응답 본문이 수신됩니다.

## 구성 요소에 대한 사용 권한 설정

Orchestrator REST API를 사용하여 구성 요소에 대한 사용 권한을 설정할 수 있습니다.

### 사전 요구 사항

설정할 수 있는 사용 권한 유형 및 요청 본문에서 사용할 수 있는 구문을 검토합니다. [REST API 사용 권한](#) 항목을 참조하십시오.

## 절차

- 1 GET 요청을 하고 반환된 구성 요소 목록에서 구성 요소의 ID를 검색합니다.

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

- 2 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용 권한을 설정할 구성 요소의 속성을 정의합니다.
- 3 요청 본문에서 설정할 사용 권한을 지정합니다.

#### 4 구성 요소 사용 권한의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

POST 요청이 성공하면 상태 코드 201이 수신됩니다. 구성 요소 사용 권한에 대한 정보가 응답 본문에 제공됩니다.

## 플러그인을 사용하여 작업 수행

Orchestrator REST API는 플러그인을 사용하여 여러 작업을 수행하는 데 사용할 수 있는 웹 서비스를 제공합니다.

### 플러그인에 대한 정보 검색

Orchestrator REST API를 사용하여 설치된 모든 플러그인에 대한 메타데이터 정보를 검색할 수 있습니다.

#### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 플러그인의 속성을 정의합니다.
- 2 플러그인 개체의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/plugins/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다.

### 플러그인 가져오기

Orchestrator REST API를 사용하여 플러그인을 가져올 수 있습니다.

REST 클라이언트 애플리케이션의 라이브러리에 따라 플러그인의 속성을 정의하는 사용자 지정 코드를 사용할 수 있습니다.

---

**참고** 이름이 같은 플러그인이 이미 설치된 경우에는 플러그인을 가져올 수 없습니다.

---

#### 사전 요구 사항

플러그인 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

#### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 가져올 플러그인의 속성을 정의합니다.
- 2 플러그인 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/plugins/
```

POST 요청이 성공하면 상태 코드 200이 수신됩니다.

## 플러그인 내보내기

Orchestrator REST API를 사용하여 플러그인을 내보낼 수 있습니다.

### 절차

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.
  - 이름: accept
  - 값: application/dar
- 2 내보내려는 플러그인의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/plugins/{plug-in_name}/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 플러그인 콘텐츠가 응답 본문에 제공됩니다.

## 플러그인을 사용하거나 사용하지 않도록 설정

Orchestrator REST API를 사용하여 플러그인을 사용하거나 사용하지 않도록 설정할 수 있습니다.

플러그인의 URL에서 PUT 요청을 만들어 플러그인의 상태를 사용에서 사용 안 함 또는 사용 안 함에서 사용으로 변경할 수 있습니다. Orchestrator 플러그인에 대한 정보를 검색하여 플러그인의 현재 상태를 확인할 수 있습니다. [플러그인에 대한 정보 검색](#) 항목을 참조하십시오.

### 사전 요구 사항

플러그인 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

### 절차

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 사용하거나 사용하지 않도록 설정할 플러그인의 속성을 정의합니다.
- 2 사용하거나 사용하지 않도록 설정할 플러그인의 URL에서 PUT 요청을 만듭니다.

```
PUT http://{orchestrator_host}:{port}/vco/api/plugins/{plug-in_name}/state/
```

PUT 요청이 성공하면 상태 코드 200이 수신됩니다.

## 서버 구성 작업 수행

Orchestrator REST API는 Orchestrator 서버 구성과 관련된 여러 작업을 수행하는 데 사용할 수 있는 웹 서비스를 제공합니다.

## Orchestrator 서버 구성에 대한 정보 검색

Orchestrator REST API를 사용하여 Orchestrator 서버 구성에 대한 정보를 검색할 수 있습니다.

**절차**

- 1 REST 클라이언트 애플리케이션에서 요청 헤더를 추가하여 정보를 검색할 서버의 속성을 정의합니다.
- 2 플러그인 개체의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다.

**Orchestrator 서버 구성 가져오기**

Orchestrator REST API를 사용하여 저장된 구성을 가져올 수 있습니다.

**사전 요구 사항**

구성 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

**절차**

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.
  - 이름: content-type
  - 값: multipart/form-data
- 2 서버 구성의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

POST 요청이 성공하면 상태 코드 200이 수신됩니다.

**Orchestrator 서버 구성 내보내기**

Orchestrator REST API를 사용하여 서버 구성을 내보낼 수 있습니다.

**사전 요구 사항**

구성 이진 콘텐츠를 다중 파트 콘텐츠로 사용할 수 있어야 합니다. 자세한 내용은 RFC 2387을 참조하십시오.

**절차**

- 1 REST 클라이언트 애플리케이션에서 다음 값을 사용하여 요청 헤더를 추가합니다.
  - 이름: content-type
  - 값: multipart/form-data
- 2 다음 값을 사용하여 또 다른 요청 헤더를 추가합니다.
  - 이름: accept

- 값: \*/\*

### 3 서버 구성의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

POST 요청이 성공하면 상태 코드 200이 수신됩니다.

## 태그 지정 작업 수행

Orchestrator REST API는 Orchestrator에서 태그를 사용하여 개체를 보다 쉽게 검색할 수 있도록 하기 위해 여러 작업을 수행하는 데 사용할 수 있는 웹 서비스를 제공합니다.

태그를 첨부하여 개체를 보다 쉽게 검색할 수 있도록 할 수 있습니다. 태그는 3~64자 길이의 문자열이며, 공백 문자를 포함할 수 없습니다.

전역 및 전용 태그를 추가할 수 있습니다. 전역 태그는 모든 Orchestrator 사용자에게 표시되고, 전용 태그는 해당 태그를 만든 사용자에게만 표시됩니다. 관리 권한이 있는 사용자만 전역 태그를 만들고 제거할 수 있습니다.

## 개체 태그 지정

Orchestrator REST API를 사용하여 개체에 태그를 할당할 수 있습니다.

전용 태그와 전역 태그를 모두 만들 수 있습니다. 요청 본문에서 태그가 전용인지 전역인지 지정합니다.

---

**참고** 글로벌 태그를 생성하려면 관리 권한이 있는 사용자로 로그인해야 합니다.

---

생성한 태그에 값을 할당할 수도 있습니다. 이 값은 태그를 필터링하는 데 사용할 수 있는 선택적 매개 변수입니다.

### 절차

#### 1 다음 구문을 사용하여 요청 본문을 정의합니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tag-instance xmlns="http://www.vmware.com/vco" global="false">
  <name>tag_name</name>
  <value>tag_value</value>
</tag-instance>
```

---

**참고** global 변수를 "true"로 설정하여 전역 태그를 만들 수 있습니다.

---

#### 2 개체의 URL에서 POST 요청을 만듭니다.

```
POST http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tags
```

POST 요청이 성공하면 상태 코드 200이 수신됩니다.

## 개체 태그 해제

Orchestrator REST API를 사용하여 개체에 할당된 태그를 제거할 수 있습니다.

전용 태그와 전역 태그를 모두 제거할 수 있습니다.

**참고** 글로벌 태그를 제거하려면 관리 권한이 있는 사용자로 로그인해야 합니다.

### 절차

- ◆ 전용 또는 전역 태그를 제거할 DELETE 요청을 만듭니다.
  - 전용 태그를 제거하려면 다음 구문을 사용하여 개체의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:
{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tag/{tag_name}
```

- 전역 태그를 제거하려면 다음 구문을 사용하여 개체의 URL에서 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tag/:
{tag_name}
```

DELETE 요청이 성공하면 상태 코드 200이 수신됩니다.

## 개체 태그 나열

Orchestrator REST API를 사용하여 개체에 할당된 태그 목록을 검색할 수 있습니다.

### 절차

- ◆ 개체의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tags
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다.

## 유형별 태그 지정 개체 나열

Orchestrator REST API를 사용하여 특정 태그가 지정된 개체 목록을 검색하고 개체 유형별로 필터링할 수 있습니다.

### 절차

- ◆ 개체 유형의 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/?tags=tag1&tags=:tag2=value
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다.

## 태그 소유자 나열

Orchestrator REST API를 사용하여 태그 소유자 목록을 검색할 수 있습니다. 태그 소유자는 하나 이상의 태그를 만든 사용자입니다.

### 절차

- ◆ 다음 URL에서 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/tags
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 하나 이상의 태그를 만든 사용자가 검색한 목록에 포함됩니다. 전역 태그는 시스템 사용자 이름 \_\_GLOBAL\_\_ 아래에 나열됩니다.

## 사용자별 태그 나열

Orchestrator REST API를 사용하여 특정 사용자가 만든 태그 목록을 검색할 수 있습니다.

또한 전역 태그를 검색할 수 있습니다. 전역 태그는 시스템 사용자 이름 \_\_GLOBAL\_\_ 아래에 나열됩니다.

### 절차

- ◆ 사용자의 URL에서 GET 요청을 만듭니다.
  - 특정 사용자가 만든 태그 목록을 검색하려면 다음 구문을 사용하여 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/tags/{user_name}
```

- 전역 태그 목록을 검색하려면 다음 구문을 사용하여 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/tags/__GLOBAL__
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다.

## 태그 이름별로 필터링된 사용자별 태그 나열

Orchestrator REST API를 사용하여 특정 사용자가 만든 태그 인스턴스 목록을 검색하고 태그 이름별로 필터링할 수 있습니다.

또한 전역 태그 인스턴스를 검색할 수 있습니다. 전역 태그는 시스템 사용자 이름 \_\_GLOBAL\_\_ 아래에 나열됩니다.

## 절차

- ◆ 사용자의 URL에서 GET 요청을 만듭니다.

- 특정 사용자가 만든 태그 인스턴스의 필터링된 목록을 검색하려면 다음 구문을 사용하여 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/tags/{user_name}/{tag_name}
```

- 전역 태그 인스턴스의 필터링된 목록을 검색하려면 다음 구문을 사용하여 GET 요청을 만듭니다.

```
GET http://{orchestrator_host}:{port}/vco/api/tags/_GLOBAL_{tag_name}
```

GET 요청이 성공하면 상태 코드 200이 수신됩니다. 태그가 지정된 개체에 대한 참조, 태그 이름, 태그 값 및 태그 인스턴스가 전역인지 전용인지 여부에 대한 표시가 검색한 정보에 포함됩니다.

## 사용자별 태그 제거

Orchestrator REST API를 사용하여 특정 사용자가 만든 모든 태그를 제거할 수 있습니다.

또한 전역 태그를 제거할 수 있습니다. 전역 태그는 시스템 사용자 이름 \_\_GLOBAL\_\_ 아래에 나열됩니다.

---

**참고** 글로벌 태그를 제거하려면 관리 권한이 있는 사용자로 로그인해야 합니다.

---

## 절차

- ◆ 사용자의 URL에서 DELETE 요청을 만듭니다.

- 특정 사용자가 만든 태그를 제거하려면 다음 구문을 사용하여 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/tags/{user_name}
```

- 전역 태그를 제거하려면 다음 구문을 사용하여 DELETE 요청을 만듭니다.

```
DELETE http://{orchestrator_host}:{port}/vco/api/tags/_GLOBAL__
```

DELETE 요청이 성공하면 상태 코드 204가 수신됩니다.