REST를 사용하여 자동화

ONTAP Select

NetApp
May 07, 2026

목차

REST를 사용하여 자동화

개념

ONTAP Select 클러스터 배포 및 관리를 위한 REST 웹 서비스 기반

REST(Representational State Transfer)는 분산 웹 애플리케이션을 생성하기 위한 스타일입니다. 웹 서비스 API 설계에 적용될 경우, 서버 기반 리소스를 노출하고 상태를 관리하기 위한 기술 및 모범 사례 세트를 설정합니다. 주요 프로토콜과 표준을 사용하여 ONTAP Select 클러스터를 배포하고 관리하기 위한 유연한 기반을 제공합니다.

아키텍처 및 기존 제약 조건

REST는 2000년 UC Irvine에서 Roy Fielding이 박사 "논문" 학위 논문에서 공식적으로 정립했습니다. 이는 일련의 제약 조건을 통해 아키텍처 스타일을 정의하며, 이러한 제약 조건들은 웹 기반 애플리케이션과 기반 프로토콜을 개선합니다. 이러한 제약 조건은 상태 비저장 통신 프로토콜을 사용하는 클라이언트/서버 아키텍처 기반의 RESTful 웹 서비스 애플리케이션을 구축합니다.

리소스 및 상태 표현

리소스는 웹 기반 시스템의 기본 구성 요소입니다. REST 웹 서비스 애플리케이션을 개발할 때 초기 설계 작업에는 다음이 포함됩니다.

- 시스템 또는 서버 기반 리소스 식별 모든 시스템은 리소스를 사용하고 유지 관리합니다. 리소스는 파일, 비즈니스 트랜잭션, 프로세스 또는 관리 엔티티일 수 있습니다. REST 웹 서비스를 기반으로 하는 애플리케이션을 설계할 때 가장 먼저 해야 할 일 중 하나는 리소스를 식별하는 것입니다.
- 리소스 상태 및 관련 상태 연산의 정의 리소스는 항상 유한한 수의 상태 중 하나에 있습니다. 상태와 상태 변경에 사용되는 관련 연산은 명확하게 정의되어야 합니다.

클라이언트와 서버 간에 메시지가 교환되어 일반적인 CRUD(Create, Read, Update, Delete) 모델에 따라 리소스 상태에 액세스하고 변경합니다.

URI 엔드포인트

모든 REST 리소스는 잘 정의된 주소 지정 체계를 사용하여 정의되고 사용 가능해야 합니다. 리소스가 위치하고 식별되는 엔드포인트는 URI(Uniform Resource Identifier)를 사용합니다. URI는 네트워크의 각 리소스에 고유한 이름을 생성하기 위한 일반적인 프레임워크를 제공합니다. URL(Uniform Resource Locator)은 웹 서비스에서 리소스를 식별하고 액세스하는 데 사용되는 URI의 한 유형입니다. 리소스는 일반적으로 파일 디렉터리와 유사한 계층 구조로 노출됩니다.

HTTP 메시지

HTTP(Hypertext Transfer Protocol)는 웹 서비스 클라이언트와 서버가 리소스에 대한 요청 및 응답 메시지를 교환하는 데 사용되는 프로토콜입니다. 웹 서비스 애플리케이션을 설계할 때 GET 및 POST와 같은 HTTP 동사는 리소스 및 해당 상태 관리 작업에 매핑됩니다.

HTTP는 상태를 저장하지 않습니다. 따라서 관련된 요청과 응답들을 하나의 트랜잭션으로 묶으려면 요청/응답 데이터 흐름과 함께 전달되는 HTTP 헤더에 추가 정보를 포함해야 합니다.

JSON 형식

클라이언트와 서버 간에 정보를 구조화하고 전송하는 방법은 여러 가지가 있지만, 가장 일반적인 방법(그리고 Deploy REST API에서 사용되는 방법)은 JavaScript Object Notation(JSON)입니다. JSON은 간단한 데이터 구조를 일반 텍스트로 표현하기 위한 업계 표준이며, 리소스를 설명하는 상태 정보를 전송하는 데 사용됩니다.

ONTAP Select Deploy API에 액세스하는 방법

REST 웹 서비스의 고유한 유연성 덕분에 ONTAP Select Deploy API는 여러 가지 방식으로 액세스할 수 있습니다.

ONTAP Select Deploy에 포함된 REST API에는 버전 번호가 할당됩니다. API 버전 번호는 Deploy 릴리스 번호와는 무관합니다. ONTAP Select 9.17.1 Deploy 관리 유틸리티에는 REST API 버전 3이 포함되어 있습니다.

유틸리티 네이티브 사용자 인터페이스 배포

API에 액세스하는 주요 방법은 ONTAP Select Deploy 웹 사용자 인터페이스를 통하는 것입니다. 브라우저는 API를 호출하고 사용자 인터페이스의 설계에 따라 데이터를 재구성합니다. 또한 Deploy 유틸리티 명령줄 인터페이스를 통해서도 API에 액세스할 수 있습니다.

ONTAP Select Deploy 온라인 설명서 페이지

ONTAP Select Deploy 온라인 문서 페이지는 브라우저를 사용할 때 또 다른 접근 경로를 제공합니다. 이 페이지는 개별 API 호출을 직접 실행하는 방법 외에도 각 호출에 대한 입력 매개변수 및 기타 옵션을 포함한 API에 대한 자세한 설명을 제공합니다. API 호출은 여러 기능 영역 또는 범주로 구성되어 있습니다.

맞춤형 프로그램

여러 가지 프로그래밍 언어 및 도구를 사용하여 Deploy API에 액세스할 수 있습니다. 널리 사용되는 언어로는 Python, Java, cURL 등이 있습니다. API를 사용하는 프로그램, 스크립트 또는 도구는 REST 웹 서비스 클라이언트 역할을 합니다. 프로그래밍 언어를 사용하면 API를 더 잘 이해하고 ONTAP Select 배포를 자동화할 수 있습니다.

ONTAP Select Deploy API 기본 작동 특성

REST는 공통된 기술과 모범 사례를 제시하지만, 각 API의 세부 사항은 설계 선택에 따라 달라질 수 있습니다. ONTAP Select Deploy API를 사용하기 전에 해당 API의 세부 사항과 운영 특성을 숙지해야 합니다.

하이퍼바이저 호스트와 ONTAP Select 노드

_하이퍼바이저 호스트_는 ONTAP Select 가상 머신을 호스팅하는 핵심 하드웨어 플랫폼입니다. ONTAP Select 가상 머신이 하이퍼바이저 호스트에 배포되어 활성화되면 해당 가상 머신은 _ONTAP Select 노드_로 간주됩니다. 배포 REST API 버전 3부터는 호스트와 노드 객체가 분리되어 있습니다. 이를 통해 하나 이상의 ONTAP Select 노드가 동일한 하이퍼바이저 호스트에서 실행될 수 있는 일대다 관계가 가능해집니다.

객체 식별자

각 리소스 인스턴스 또는 객체는 생성될 때 고유 식별자가 할당됩니다. 이러한 식별자는 ONTAP Select Deploy의 특정 인스턴스 내에서 전역적으로 고유합니다. 새 객체 인스턴스를 생성하는 API 호출을 실행하면 관련 ID 값이 HTTP

응답의 `location` 헤더에 포함되어 호출자에게 반환됩니다. 이 식별자를 추출하여 이후 호출에서 해당 리소스 인스턴스를 참조할 때 사용할 수 있습니다.

> 객체 식별자의 내용 및 내부 구조는 언제든지 변경될 수 있습니다. 따라서 관련 객체를 참조할 때는 필요에 따라 해당 API 호출에서만 식별자를 사용해야 합니다.

요청 식별자

모든 성공적인 API 요청에는 고유 식별자가 할당됩니다. 이 식별자는 관련 HTTP 응답의 `request-id` 헤더에 반환됩니다. 요청 식별자를 사용하여 특정 API 요청-응답 트랜잭션의 모든 활동을 총칭하여 참조할 수 있습니다. 예를 들어, 요청 ID를 기반으로 특정 트랜잭션에 대한 모든 이벤트 메시지를 검색할 수 있습니다.

동기식 호출과 비동기식 호출

서버가 클라이언트로부터 받은 HTTP 요청을 수행하는 주요 방법은 두 가지입니다.

- 동기식 서버가 요청을 즉시 수행하고 상태 코드 200, 201 또는 204로 응답합니다.
- 비동기 방식입니다. 서버는 요청을 수락하고 상태 코드 202로 응답합니다. 이는 서버가 클라이언트의 요청을 수락하고 요청 처리를 위한 백그라운드 작업을 시작했음을 나타냅니다. 최종 성공 또는 실패 여부는 즉시 확인할 수 없으며, 추가 API 호출을 통해 확인해야 합니다.

장기 실행 작업의 완료를 확인합니다

일반적으로 완료하는 데 시간이 오래 걸릴 수 있는 모든 작업은 서버에서 백그라운드 작업을 사용하여 비동기적으로 처리됩니다. Deploy REST API에서는 모든 백그라운드 작업에 Job 객체가 연결되어 있으며, 이 객체는 작업을 추적하고 현재 상태와 같은 정보를 제공합니다. 백그라운드 작업이 생성되면 고유 식별자를 포함한 Job 객체가 HTTP 응답으로 반환됩니다.

Job 객체를 직접 조회하여 관련 API 호출의 성공 또는 실패 여부를 확인할 수 있습니다. 자세한 내용은 _Job 객체를 사용한 비동기 처리_를 참조하십시오.

Job 객체를 사용하는 것 외에도 다음과 같은 방법으로 요청의 성공 또는 실패 여부를 판단할 수 있습니다.

- 이벤트 메시지 특정 API 호출과 관련된 모든 이벤트 메시지는 원래 응답에 포함된 요청 ID를 사용하여 검색할 수 있습니다. 이벤트 메시지에는 일반적으로 성공 또는 실패 여부가 표시되며 오류 상황을 디버깅할 때도 유용하게 사용할 수 있습니다.
- 리소스 상태 여러 리소스는 상태 값을 유지하며, 이 값을 조회하여 요청의 성공 또는 실패 여부를 간접적으로 확인할 수 있습니다.

보안

Deploy API는 다음과 같은 보안 기술을 사용합니다.

- 전송 계층 보안 Deploy 서버와 클라이언트 간에 네트워크를 통해 전송되는 모든 트래픽은 TLS를 통해 암호화됩니다. 암호화되지 않은 채널을 통한 HTTP 프로토콜 사용은 지원되지 않습니다. TLS 버전 1.2가 지원됩니다.
- HTTP 인증 기본 인증은 모든 API 트랜잭션에 사용됩니다. base64 문자열로 사용자 이름과 암호를 포함하는 HTTP 헤더가 모든 요청에 추가됩니다.

ONTAP Select에 대한 요청 및 응답 API 트랜잭션

모든 Deploy API 호출은 Deploy 가상 머신에 대한 HTTP 요청으로 수행되며, 가상 머신은 이에 대한 응답을 클라이언트로 전송합니다. 이러한 요청/응답 쌍을 API 트랜잭션이라고 합니다. Deploy API를 사용하기 전에 요청을 제어하는 데 사용할 수 있는 입력 변수와 응답 출력 내용을 숙지해야 합니다.

API 요청을 제어하는 입력 변수

HTTP 요청에 설정된 매개변수를 통해 API 호출 처리 방식을 제어할 수 있습니다.

요청 헤더

다음을 포함한 여러 헤더를 HTTP 요청에 포함해야 합니다.

- content-type 요청 본문에 JSON이 포함된 경우 이 헤더는 application/json으로 설정해야 합니다.
- accept 응답 본문에 JSON이 포함될 경우, 이 헤더는 application/json으로 설정해야 합니다.
- authorization 기본 인증은 base64 문자열로 인코딩된 사용자 이름과 비밀번호로 설정해야 합니다.

요청 본문

요청 본문의 내용은 특정 호출에 따라 다릅니다. HTTP 요청 본문은 다음 중 하나로 구성됩니다.

- 입력 변수(예: 새 클러스터의 이름)가 포함된 JSON 객체
- 비어 있음

객체 필터링

GET을 사용하는 API 호출을 실행할 때 모든 속성을 기반으로 반환된 객체를 제한하거나 필터링할 수 있습니다. 예를 들어 일치시킬 정확한 값을 지정할 수 있습니다.

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

정확히 일치하는 항목 외에도, 특정 값 범위에 걸쳐 객체 집합을 반환하는 데 사용할 수 있는 다른 연산자가 있습니다. ONTAP Select는 아래에 표시된 필터링 연산자를 지원합니다.

운영자	설명
=	같음
<	미만
>	보다 큼
⇐	이하
>=	크거나 같음
|	또는
!	같지 않음
*	탐욕적 와일드카드

쿼리의 일부로 null 키워드 또는 그 부정형(!null)을 사용하면 특정 필드가 설정되었는지 여부에 따라 객체 집합을 반환할 수도 있습니다.

객체 필드 선택

기본적으로 GET을 사용하여 API 호출을 실행하면 객체를 고유하게 식별하는 속성만 반환됩니다. 이러한 최소 필드 집합은 각 객체의 키 역할을 하며 객체 유형에 따라 다릅니다. 다음과 같은 방법으로 fields 쿼리 매개변수를 사용하여 추가 객체 속성을 선택할 수 있습니다.

- 비용이 저렴한 필드 `fields=*`를 지정하여 로컬 서버 메모리에 유지되거나 액세스하는 데 처리량이 거의 필요하지 않은 객체 필드를 검색합니다.
- 비용이 많이 드는 필드 `fields=**`를 지정하면 액세스하는 데 추가 서버 처리가 필요한 필드를 포함하여 모든 객체 필드를 검색합니다.
- 사용자 지정 필드 선택 `fields=FIELDNAME`을 사용하여 원하는 필드를 정확하게 지정할 수 있습니다. 여러 필드를 요청할 경우 값은 공백 없이 쉼표로 구분해야 합니다.

모범 사례로, 항상 원하는 특정 필드를 식별해야 합니다. 필요할 때만 저렴한 필드 또는 비싼 필드 세트를 가져오십시오. 저렴한 필드와 비싼 필드 분류는 NetApp의 내부 성능 분석을 기반으로 결정됩니다. 특정 필드에 대한 분류는 언제든지 변경될 수 있습니다.

출력 세트의 객체를 정렬합니다

리소스 컬렉션의 레코드는 객체에 정의된 기본 순서대로 반환됩니다. 다음과 같이 필드 이름과 정렬 방향을 지정하여 order_by 쿼리 매개변수를 사용하면 순서를 변경할 수 있습니다:

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

예를 들어, type 필드를 내림차순으로 정렬한 다음 id 필드를 오름차순으로 정렬할 수 있습니다.

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

여러 매개변수를 포함할 경우 필드를 쉼표로 구분해야 합니다.

페이지 매김

동일한 유형의 객체 컬렉션에 접근하기 위해 GET 메서드를 사용하는 API 호출 시, 기본적으로 일치하는 모든 객체가 반환됩니다. 필요한 경우, 요청에 max_records 쿼리 매개변수를 사용하여 반환되는 레코드 수를 제한할 수 있습니다. 예를 들면 다음과 같습니다.

```
max_records=20
```

필요한 경우 이 매개변수를 다른 쿼리 매개변수와 결합하여 결과 집합을 좁힐 수 있습니다. 예를 들어 다음은 지정된 시간 이후에 생성된 시스템 이벤트를 최대 10개까지 반환합니다:

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

이벤트(또는 모든 객체 유형)를 페이지별로 보기 위해 여러 번 요청을 보낼 수 있습니다. 각 후속 API 호출은 이전 결과 집합의 최신 이벤트를 기준으로 새로운 시간 값을 사용해야 합니다.

API 응답 해석

각 API 요청은 클라이언트로 응답을 반환합니다. 응답을 검토하여 요청이 성공했는지 여부를 확인하고 필요에 따라 추가 데이터를 가져올 수 있습니다.

HTTP 상태 코드

Deploy REST API에서 사용하는 HTTP 상태 코드는 다음과 같습니다.

코드	의미	설명
200	OK	새 객체를 생성하지 않는 호출이 성공했음을 나타냅니다.
201	생성됨	객체가 성공적으로 생성되었습니다. 위치 응답 헤더에 해당 객체의 고유 식별자가 포함되어 있습니다.
202	수락됨	요청을 처리하기 위해 장시간 실행되는 백그라운드 작업이 시작되었지만, 아직 작업이 완료되지 않았습니다.
400	잘못된 요청	요청 입력값이 인식되지 않거나 부적절합니다.
403	금지됨	권한 오류로 인해 액세스가 거부되었습니다.
404	찾을 수 없음	요청에서 참조한 리소스가 존재하지 않습니다.
405	메서드가 허용되지 않습니다	요청에 사용된 HTTP 동사는 해당 리소스에서 지원되지 않습니다.
409	충돌	객체가 이미 존재하므로 객체를 생성하지 못했습니다.
500	내부 오류	서버에서 일반적인 내부 오류가 발생했습니다.
501	구현되지 않음	URI는 알려져 있지만 요청을 수행할 수 없습니다.

응답 헤더

Deploy 서버에서 생성되는 HTTP 응답에는 다음과 같은 여러 헤더가 포함됩니다.

- request-id 모든 성공적인 API 요청에는 고유한 요청 식별자가 할당됩니다.
- location 객체가 생성될 때 location 헤더에는 고유한 객체 식별자를 포함하여 새 객체의 전체 URL이 포함됩니다.

응답 본문

API 요청과 관련된 응답의 내용은 객체, 처리 유형 및 요청의 성공 또는 실패에 따라 다릅니다. 응답 본문은 JSON으로 렌더링됩니다.

- 단일 객체 단일 객체는 요청에 따라 필드 집합과 함께 반환될 수 있습니다. 예를 들어, GET을 사용하여 고유 식별자를 통해 클러스터의 선택된 속성을 검색할 수 있습니다.
- 여러 객체 리소스 컬렉션에서 여러 객체를 반환할 수 있습니다. 모든 경우에 일관된 형식이 사용되며, `num_records`레코드 수와 객체 인스턴스 배열을 포함하는 레코드가 표시됩니다. 예를 들어 특정 클러스터에 정의된 모든 노드를 검색할 수 있습니다.
- Job 객체 API 호출이 비동기적으로 처리되는 경우 백그라운드 작업을 고정하는 Job 객체가 반환됩니다. 예를 들어, 클러스터를 배포하는 데 사용되는 POST 요청은 비동기적으로 처리되며 Job 객체를 반환합니다.
- Error 객체 오류가 발생하면 항상 Error 객체가 반환됩니다. 예를 들어, 이미 존재하는 이름으로 클러스터를 생성하려고 하면 오류가 발생합니다.
- 비어 있음 특정 경우에는 데이터가 반환되지 않고 응답 본문이 비어 있습니다. 예를 들어, DELETE를 사용하여 기존 호스트를 삭제한 후에는 응답 본문이 비어 있습니다.

ONTAP Select용 Job 객체를 사용한 비동기 처리

Deploy API 호출 중 일부, 특히 리소스를 생성하거나 수정하는 호출은 다른 호출보다 완료하는 데 시간이 더 오래 걸릴 수 있습니다. ONTAP Select Deploy는 이러한 장시간 실행되는 요청을 비동기적으로 처리합니다.

Job 객체를 사용하여 설명된 비동기 요청

비동기적으로 실행되는 API 호출 후 HTTP 응답 코드 202는 요청이 성공적으로 검증되고 수락되었지만 아직 완료되지 않았음을 나타냅니다. 요청은 백그라운드 작업으로 처리되며, 클라이언트에 대한 초기 HTTP 응답 이후에도 계속 실행됩니다. 응답에는 요청을 연결하는 Job 객체와 해당 고유 식별자가 포함됩니다.

비동기적으로 작동하는 API 호출을 확인하려면 ONTAP Select Deploy 온라인 설명서 페이지를 참조하십시오.

API 요청과 연결된 Job 객체를 쿼리합니다

HTTP 응답에 반환되는 Job 객체에는 여러 속성이 포함되어 있습니다. state 속성을 조회하여 요청이 성공적으로 완료되었는지 확인할 수 있습니다. Job 객체는 다음 상태 중 하나일 수 있습니다.

- 대기 중
- 실행 중
- 성공
- 실패

Job 객체를 폴링하여 작업의 종료 상태(성공 또는 실패)를 감지하는 데 사용할 수 있는 두 가지 기술이 있습니다.

- 표준 폴링 요청 현재 작업 상태가 즉시 반환됩니다
- 장기 폴링 요청 작업 상태는 다음 중 하나가 발생할 때만 반환됩니다.
  - 상태가 폴 요청에 제공된 날짜-시간 값보다 최근에 변경되었습니다
  - 타임아웃 값이 만료되었습니다(1~120초)

표준 폴링과 롱 폴링은 동일한 API 호출을 사용하여 Job 객체를 쿼리합니다. 그러나 롱 폴링 요청에는 두 개의 쿼리 매개변수 `poll_timeout`및 `last_modified`가 포함됩니다.

Deploy 가상 머신의 작업 부하를 줄이려면 항상 롱 폴링을 사용해야 합니다.

비동기 요청을 발행하는 일반적인 절차

다음과 같은 상위 수준 절차를 사용하여 비동기 API 호출을 완료할 수 있습니다.

1. 비동기 API 호출을 실행합니다.
2. 요청이 성공적으로 수락되었음을 나타내는 HTTP 응답 202를 수신합니다.
3. 응답 본문에서 Job 객체의 식별자를 추출합니다.
4. 루프 내에서 각 주기마다 다음을 수행합니다.

a. 롱폴 요청을 사용하여 작업의 현재 상태를 가져옵니다
b. 작업이 비종료 상태(대기 중, 실행 중)인 경우 루프를 다시 수행합니다.

5. 작업이 최종 상태(성공, 실패)에 도달하면 중지합니다.

브라우저로 액세스

브라우저를 통해 **ONTAP Select Deploy API**에 액세스하기 전에

Deploy 온라인 설명서 페이지를 사용하기 전에 알아두어야 할 몇 가지 사항이 있습니다.

배포 계획

특정 배포 또는 관리 작업을 수행하는 과정에서 API 호출을 사용하려는 경우 배포 계획을 수립하는 것이 좋습니다. 이러한 계획은 공식적이거나 비공식적일 수 있으며, 일반적으로 목표와 사용할 API 호출을 포함합니다. 자세한 내용은 Deploy REST API를 사용하는 워크플로 프로세스를 참조하십시오.

JSON 예제 및 매개변수 정의

각 API 호출은 일관된 형식을 사용하여 문서 페이지에 설명되어 있습니다. 내용에는 구현 참고 사항, 쿼리 매개변수 및 HTTP 상태 코드가 포함됩니다. 또한 다음과 같이 API 요청 및 응답에 사용된 JSON에 대한 세부 정보를 표시할 수 있습니다.

- 예시 값 API 호출에서 _Example Value_를 클릭하면 해당 호출에 대한 일반적인 JSON 구조가 표시됩니다. 필요에 따라 예시를 수정하여 요청 입력값으로 사용할 수 있습니다.
- Model을 클릭하면 각 매개변수에 대한 설명과 함께 JSON 매개변수의 전체 목록이 표시됩니다.

API 호출 시 주의하십시오

Deploy 문서 페이지를 사용하여 수행하는 모든 API 작업은 실시간 작업입니다. 구성 또는 기타 데이터를 실수로 생성, 업데이트 또는 삭제하지 않도록 주의해야 합니다.

ONTAP Select Deploy 문서 페이지에 액세스합니다

API 문서를 표시하고 수동으로 API 호출을 실행하려면 ONTAP Select Deploy 온라인 문서 페이지에 액세스해야 합니다.

시작하기 전에

다음 조건을 충족해야 합니다.

- ONTAP Select Deploy 가상 머신의 IP 주소 또는 도메인 이름
- 관리자의 사용자 이름 및 암호

단계

1. 브라우저에 URL을 입력하고 **Enter** 키를 누르십시오.

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

2. 관리자 사용자 이름과 비밀번호를 사용하여 Sign in하십시오.

결과

Deploy 문서 웹 페이지는 페이지 하단에 범주별로 정리된 호출 목록과 함께 표시됩니다.

ONTAP Select Deploy API 호출을 이해하고 실행합니다

모든 API 호출에 대한 세부 정보는 ONTAP Select Deploy 온라인 문서 웹 페이지에서 공통 형식으로 문서화되어 표시됩니다. 하나의 API 호출을 이해하면 모든 API 호출의 세부 정보에 액세스하고 해석할 수 있습니다.

시작하기 전에

ONTAP Select Deploy 온라인 설명서 웹 페이지에 Sign in해야 합니다. 클러스터 생성 시 ONTAP Select 클러스터에 할당된 고유 식별자가 있어야 합니다.

이 작업 정보

고유 식별자를 사용하여 ONTAP Select 클러스터를 설명하는 구성 정보를 검색할 수 있습니다. 이 예에서는 비용이 저렴한 것으로 분류된 모든 필드가 반환됩니다. 하지만 최적의 결과를 얻으려면 필요한 특정 필드만 요청하는 것이 좋습니다.

단계

1. 기본 페이지 맨 아래로 스크롤하여 *Cluster*를 클릭합니다.
2. *GET /clusters/{cluster_id}*를 클릭하여 ONTAP Select 클러스터에 대한 정보를 반환하는 데 사용된 API 호출의 세부 정보를 표시합니다.

워크플로 프로세스

ONTAP Select Deploy API 워크플로를 사용하기 전에

워크플로 프로세스를 검토하고 사용할 준비를 해야 합니다.

워크플로에서 사용되는 **API** 호출 이해

ONTAP Select 온라인 문서 페이지에는 모든 REST API 호출에 대한 세부 정보가 포함되어 있습니다. 여기서는 해당 세부 정보를 반복하지 않고, 워크플로 샘플에서 사용되는 각 API 호출에는 문서 페이지에서 해당 호출을 찾는 데 필요한 정보만 포함되어 있습니다. 특정 API 호출을 찾은 후에는 입력 매개변수, 출력 형식, HTTP 상태 코드 및 요청 처리 유형을 포함한 호출의 전체 세부 정보를 확인할 수 있습니다.

워크플로 내의 각 API 호출에 대해 다음 정보가 포함되어 있어 문서 페이지에서 해당 호출을 쉽게 찾을 수 있습니다.

- 범주 API 호출은 문서 페이지에서 기능적으로 관련된 영역 또는 범주로 정리되어 있습니다. 특정 API 호출을 찾으려면 페이지 하단으로 스크롤하여 해당 API 범주를 클릭하십시오.
- HTTP 동사 HTTP 동사는 리소스에 대해 수행되는 작업을 식별합니다. 각 API 호출은 하나의 HTTP 동사를 통해 실행됩니다.
- 경로 경로는 호출 수행의 일부로 작업이 적용되는 특정 리소스를 결정합니다. 경로 문자열은 핵심 URL에 추가되어 리소스를 식별하는 전체 URL을 구성합니다.

REST API에 직접 액세스할 수 있는 URL 구성

ONTAP Select 설명서 페이지 외에도 Python과 같은 프로그래밍 언어를 통해 Deploy REST API에 직접 액세스할 수 있습니다. 이 경우 핵심 URL은 온라인 설명서 페이지에 액세스할 때 사용하는 URL과 약간 다릅니다. API에 직접 액세스할 때는 도메인과 포트 문자열 뒤에 /api를 추가해야 합니다. 예를 들면 다음과 같습니다.
`http://deploy.mycompany.com/api`

워크플로 1: ESXi에 ONTAP Select 단일 노드 평가 클러스터 생성

vCenter에서 관리하는 VMware ESXi 호스트에 단일 노드 ONTAP Select 클러스터를 구축할 수 있습니다. 클러스터는 평가판 라이선스로 생성됩니다.

클러스터 생성 워크플로는 다음과 같은 상황에서 다릅니다.

- ESXi 호스트는 vCenter(독립 실행형 호스트)에서 관리되지 않습니다
- 클러스터 내에서 여러 노드 또는 호스트가 사용됩니다.
- 클러스터는 구매한 라이센스를 사용하여 운영 환경에 배포됩니다.
- VMware ESXi 대신 KVM 하이퍼바이저가 사용됩니다

1. vCenter 서버 자격 증명 등록

vCenter 서버에서 관리하는 ESXi 호스트에 배포할 때는 호스트를 등록하기 전에 자격 증명을 추가해야 합니다. 그러면 Deploy 관리 유틸리티가 해당 자격 증명을 사용하여 vCenter에 인증할 수 있습니다.

범주	HTTP 동사	경로
배포	POST	/security/credentials

Curl

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

JSON 입력(step01)

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

처리 유형
  비동기

출력

- 위치 응답 헤더의 자격 증명 ID
- 작업 객체

2. 하이퍼바이저 호스트를 등록합니다

ONTAP Select 노드가 포함된 가상 머신이 실행될 하이퍼바이저 호스트를 추가해야 합니다.

범주	**HTTP** 동사	경로
클러스터	POST	/hosts

Curl

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

JSON 입력**(step02)**

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

처리 유형

비동기

출력

- 위치 응답 헤더의 호스트 ID
- 작업 객체

3. 클러스터 생성

ONTAP Select 클러스터를 생성하면 기본 클러스터 구성이 등록되고 노드 이름은 Deploy에서 자동으로 생성됩니다.

범주	**HTTP** 동사	경로
클러스터	POST	/clusters

Curl

단일 노드 클러스터의 경우 쿼리 매개 변수 node_count를 1로 설정해야 합니다.

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

JSON 입력(step03)

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

처리 유형

동기식

출력

- 위치 응답 헤더의 클러스터 ID

4. 클러스터를 구성합니다

클러스터 구성 시 제공해야 하는 몇 가지 속성이 있습니다.

범주	**HTTP** 동사	경로
클러스터	PATCH	/클러스터/{cluster_id}

Curl

클러스터 ID를 제공해야 합니다.

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

JSON 입력(step04)

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

처리 유형

동기식

출력

None

5. 노드 이름 검색

Deploy 관리 유틸리티는 클러스터를 생성할 때 노드 식별자와 이름을 자동으로 생성합니다. 노드를 구성하기 전에 할당된 ID를 가져와야 합니다.

범주	**HTTP** 동사	경로
클러스터	GET	/clusters/{cluster_id}/nodes

Curl

클러스터 ID를 제공해야 합니다.

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

처리 유형

동기식

출력

- 고유 ID와 이름을 가진 단일 노드를 각각 설명하는 배열 레코드

6. 노드를 구성합니다

노드를 구성하는 데 사용되는 세 가지 API 호출 중 첫 번째인 노드의 기본 구성을 제공해야 합니다.

범주	**HTTP** 동사	경로
클러스터	PATH	/클러스터/{cluster_id}/노드/{node_id}

Curl

클러스터 ID와 노드 ID를 제공해야 합니다.

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

JSON 입력**(step06)**

ONTAP Select 노드가 실행될 호스트 ID를 제공해야 합니다.

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

처리 유형
동기식

출력
None

7. 노드 네트워크를 검색합니다

단일 노드 클러스터의 노드에서 사용하는 데이터 및 관리 네트워크를 식별해야 합니다. 내부 네트워크는 단일 노드 클러스터에서 사용되지 않습니다.

범주	**HTTP** 동사	경로
클러스터	GET	/클러스터/{cluster_id}/노드/{node_id}/네트워크

Curl
클러스터 ID와 노드 ID를 제공해야 합니다.

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

처리 유형
동기식

출력
- 고유 ID 및 용도를 포함하여 노드의 단일 네트워크를 각각 설명하는 두 개의 레코드 배열

8. 노드 네트워킹을 구성합니다

데이터 및 관리 네트워크를 구성해야 합니다. 내부 네트워크는 단일 노드 클러스터에서 사용되지 않습니다.

다음 API 호출을 네트워크별로 한 번씩, 총 두 번 실행하십시오.

범주	**HTTP** 동사	경로
클러스터	PATCH	/클러스터/{cluster_id}/노드/{node_id}/네트워크/{network_id}

Curl

클러스터 ID, 노드 ID 및 네트워크 ID를 제공해야 합니다.

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

JSON 입력(step08)

네트워크 이름을 제공해야 합니다.

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

처리 유형

동기식

출력

None

9. 노드 스토리지 풀 구성

노드 구성의 마지막 단계는 스토리지 풀을 연결하는 것입니다. vSphere 웹 클라이언트를 통해 또는 선택적으로 Deploy REST API를 통해 사용 가능한 스토리지 풀을 확인할 수 있습니다.

범주	**HTTP** 동사	경로
클러스터	PATCH	/클러스터/{cluster_id}/노드/{node_id}/네트워크/{network_id}

Curl

클러스터 ID, 노드 ID 및 네트워크 ID를 제공해야 합니다.

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

JSON 입력(step09)

풀 용량은 2TB입니다.

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

처리 유형

동기식

출력

None

10. 클러스터를 배포합니다

클러스터와 노드 구성이 완료되면 클러스터를 배포할 수 있습니다.

범주	**HTTP** 동사	경로
클러스터	POST	/clusters/{cluster_id}/deploy

Curl

클러스터 ID를 제공해야 합니다.

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

JSON 입력**(10**단계**)**

ONTAP 관리자 계정의 암호를 제공해야 합니다.

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

처리 유형

비동기

출력

- 작업 객체

관련 정보

"ONTAP Select 클러스터의 90일 평가판 인스턴스 배포"

Python을 이용한 액세스

Python을 사용하여 ONTAP Select Deploy API에 액세스하기 전에

샘플 Python 스크립트를 실행하기 전에 환경을 준비해야 합니다.

Python 스크립트를 실행하기 전에 환경이 올바르게 구성되었는지 확인해야 합니다.

- 최신 버전의 Python2가 설치되어 있어야 합니다. 샘플 코드는 Python2를 사용하여 테스트되었습니다. Python3에서도 사용 가능할 것으로 예상되지만, 호환성 테스트는 수행되지 않았습니다.
- Requests 및 urllib3 라이브러리가 설치되어 있어야 합니다. 사용 환경에 따라 pip 또는 다른 Python 관리 툴을 사용할 수 있습니다.
- 스크립트가 실행되는 클라이언트 워크스테이션은 ONTAP Select Deploy 가상 머신에 대한 네트워크 액세스 권한이 있어야 합니다.

또한 다음 정보가 있어야 합니다.

- Deploy 가상 머신의 IP 주소
- Deploy 관리자 계정의 사용자 이름과 비밀번호

ONTAP Select Deploy용 Python 스크립트 이해

샘플 Python 스크립트를 사용하면 여러 가지 작업을 수행할 수 있습니다. 실제 Deploy 인스턴스에서 사용하기 전에 스크립트를 이해해야 합니다.

일반적인 설계 특성

이 스크립트는 다음과 같은 공통 특성을 갖도록 설계되었습니다.

- 클라이언트 시스템의 명령줄 인터페이스에서 실행 적절하게 구성된 클라이언트 시스템에서 Python 스크립트를 실행할 수 있습니다. 자세한 내용은 _시작하기 전에_를 참조하십시오.
- CLI 입력 매개 변수 수락 각 스크립트는 입력 매개 변수를 통해 CLI에서 제어됩니다.
- 입력 파일 읽기 각 스크립트는 목적에 따라 입력 파일을 읽습니다. 클러스터를 생성하거나 삭제할 때는 JSON 구성 파일을 제공해야 합니다. 노드 라이센스를 추가할 때는 유효한 라이센스 파일을 제공해야 합니다.
- 공통 지원 모듈 사용 공통 지원 모듈 _deploy_requests.py_에는 단일 클래스가 포함되어 있습니다. 이 클래스는 각 스크립트에서 가져와 사용됩니다.

클러스터 생성

cluster.py 스크립트를 사용하여 ONTAP Select 클러스터를 생성할 수 있습니다. CLI 매개변수와 JSON 입력 파일의 내용을 기반으로 배포 환경에 맞게 스크립트를 다음과 같이 수정할 수 있습니다.

- 하이퍼바이저 배포는 ESXi 또는 KVM(배포 릴리스에 따라 다름)에 할 수 있습니다. ESXi에 배포하는 경우 하이퍼바이저는 vCenter에서 관리하거나 독립형 호스트로 구성할 수 있습니다.

- 클러스터 크기 단일 노드 또는 다중 노드 클러스터를 배포할 수 있습니다.
- 평가판 또는 프로덕션 라이센스 평가판 또는 구매한 프로덕션 라이센스를 사용하여 클러스터를 배포할 수 있습니다.

스크립트의 CLI 입력 매개변수는 다음과 같습니다.

- Deploy 서버의 호스트 이름 또는 IP 주소
- admin 사용자 계정의 암호
- JSON 구성 파일의 이름
- 메시지 출력에 대한 상세 표시 플래그

노드 라이센스 추가

프로덕션 클러스터를 배포하려면 *add_license.py* 스크립트를 사용하여 각 노드에 라이센스를 추가해야 합니다. 라이센스는 클러스터 배포 전이나 후에 추가할 수 있습니다.

스크립트의 CLI 입력 매개변수는 다음과 같습니다.

- Deploy 서버의 호스트 이름 또는 IP 주소
- admin 사용자 계정의 암호
- 라이센스 파일 이름
- 라이센스를 추가할 수 있는 권한을 가진 ONTAP 사용자 이름
- ONTAP 사용자의 암호

클러스터 삭제

delete_cluster.py 스크립트를 사용하여 기존 ONTAP Select 클러스터를 삭제할 수 있습니다.

스크립트의 CLI 입력 매개변수는 다음과 같습니다.

- Deploy 서버의 호스트 이름 또는 IP 주소
- admin 사용자 계정의 암호
- JSON 구성 파일의 이름

Python 코드 샘플

ONTAP Select 클러스터를 생성하는 스크립트

다음 스크립트를 사용하면 스크립트 내에 정의된 매개변수와 JSON 입력 파일을 기반으로 클러스터를 생성할 수 있습니다.

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: cluster.py
```

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

import traceback
import argparse
import json
import logging

from deploy_requests import DeployRequests


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

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


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

    hosts = config.get('hosts', [])
```

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


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

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

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

    missing_host_cnt = 0
    for host in config['hosts']:
        if not deploy.resource_exists('/hosts', 'name', host['name']):
            missing_host_cnt += 1
            host_config = {"name": host['name'], "hypervisor_type": host[
'type']}
            if 'mgmt_server' in host:
                host_config["management_server"] = host['mgmt_server']
                log_info(
                   "Registering from vcenter {mgmt_server}".format(**
host))

            if 'password' in host and 'user' in host:
                host_config['credential'] = {
```

```
                    "password": host['password'], "username": host['user
']}

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

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


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

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

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

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

        num_nodes = len(config['nodes'])

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

        resp = deploy.post('/v3/clusters?node_count={}'.format(num_nodes),
data)
        cluster_id = resp.headers.get('Location').split('/')[-1]

    return cluster_id


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

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


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

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

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

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

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

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

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

    log_info("Node properties: {}".format(data))
    deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id),
data)


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

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

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

```
(cluster_id))

    for network in node['networks']:

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

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

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


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

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

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

    if 'disks' in node['storage'] and node['storage']['disks']:
        data = {'disks': node['storage']['disks']}
        deploy.post(
            '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id,
node_id), data)


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

    cluster_id = add_cluster_attributes(deploy, config)
```

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

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

    return cluster_id


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

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


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


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


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


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

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

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

        add_vcenter_credentials(deploy, config)

        add_standalone_host_credentials(deploy, config)

        register_unkown_hosts(deploy, config)

        cluster_id = create_cluster_config(deploy, config)

        deploy_cluster(deploy, cluster_id, config)


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

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

ONTAP Select 클러스터를 생성하는 스크립트용 JSON

Python 코드 샘플을 사용하여 ONTAP Select 클러스터를 생성하거나 삭제할 때는 스크립트에 JSON 파일을 입력으로 제공해야 합니다. 배포 계획에 따라 적절한 JSON 샘플을 복사하여 수정할 수 있습니다.

ESXi 기반 단일 노드 클러스터

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

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

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

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",
      "name": "node-1",
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
          "vlan": 1234
        },
        {
          "name": "ontap-external",
          "purpose": "data",
          "vlan": null
        },
        {
```

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

vCenter를 사용하는 **ESXi**의 단일 노드 클러스터

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

  "cluster": {
    "dns_info": {"domains": ["lab1.company-demo.com", "lab2.company-
demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
      ],
      "dns_ips": ["10.206.80.135","10.206.80.136"]
  },

  "ontap_image_version":"9.7",
  "gateway":"10.206.80.1",
  "ip":"10.206.80.115",
  "name":"mycluster",
  "ntp_servers": ["10.206.80.183","10.206.80.142"],
```

```
  "ontap_admin_password":"mypassword2",
  "netmask":"255.255.254.0"
  },

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

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

      "host_name": "host-1234",
      "is_storage_efficiency_enabled": false,
      "instance_type": "small",
      "storage": {
        "disk":[],
        "pools": [
          {
            "name": "storage-pool-1",
            "capacity":5685190380748
          }
        ]
      }
    }
```

```
    ]
}
```

KVM의 단일 노드 클러스터

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

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

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

    "ontap_image_version": "9.7",
    "gateway":"10.206.80.1",
    "ip":"10.206.80.115",
    "name":"CBF4ED97",
    "ntp_servers": ["10.206.80.183", "10.206.80.142"],
    "ontap_admin_password": "mypassword2",
    "netmask":"255.255.254.0"
  },
  "nodes": [
    {
      "serial_number":"3200000nn",
      "ip":"10.206.80.115",
      "name": "node-1",
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
          "vlan":1234
        },
        {
          "name": "ontap-external",
```

```
          "purpose": "data",
          "vlan": null
        },
        {
          "name": "ontap-internal",
          "purpose": "internal",
          "vlan": null
        }
      ],

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

ONTAP Select 노드 라이센스를 추가하는 스크립트

다음 스크립트를 사용하여 ONTAP Select 노드에 라이센스를 추가할 수 있습니다.

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

```
# no less restrictive than those set forth herein.
#
##------------------------------------------------------------------

import argparse
import logging
import json

from deploy_requests import DeployRequests


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

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

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


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

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


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

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

    put_license(deploy, serial_number, data, files)


def put_free_license(deploy, serial_number, license_filename):
    data = {}
```

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

    put_license(deploy, serial_number, data, files)


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

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

        return serialNumber


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


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


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


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

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

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

        # If the license already exists in the Deploy server, determine if
its used
```

```
        if deploy.find_resource('/clusters', 'nodes.serial_number',
serial_number):

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


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

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

ONTAP Select 클러스터를 삭제하는 스크립트

다음 CLI 스크립트를 사용하여 기존 클러스터를 삭제할 수 있습니다.

```

```

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

import argparse
import json
import logging

from deploy_requests import DeployRequests

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


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


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

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


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


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

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

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

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

        offline_cluster(deploy, cluster_id)

        delete_cluster(deploy, cluster_id)


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

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

ONTAP Select용 공통 지원 Python 모듈

모든 Python 스크립트는 단일 모듈에 있는 공통 Python 클래스를 사용합니다.

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

import json
import logging
import requests

requests.packages.urllib3.disable_warnings()

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

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

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

```
                                    auth=self.auth, verify=False,
                                    json=data,
                                    headers=self.headers)

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

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

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

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

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

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

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

```
        return response

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

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

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

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

    def get_num_records(self, path, query=None):
        ''' Returns the number of records found in a container, or None on
error '''
        resource = None
        query_opt = '?{}'.format(query) if query else ''
        response = self.get('{path}{query}'.format(path=path, query
=query_opt))
        if response.status_code == 200 :
            return response.json().get('num_records')
        return None

    def resource_exists(self, path, name, value):
```

```
        return self.find_resource(path, name, value) is not None

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

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

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

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

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

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

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

    def exit_on_errors(self, response):
        if response.status_code >= 400:
            self.logger.error('FAILED request to URL: %s\nHEADERS: %s
\nRESPONSE BODY: %s',
                              response.request.url,
                              self.filter_headers(response),
                              response.text)
        response.raise_for_status()    # Displays the response error, and
exits the script

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

```
'request-id'] if key in response.headers}
```

ONTAP Select 클러스터 노드의 크기를 조정하는 스크립트

다음 스크립트를 사용하여 ONTAP Select 클러스터의 노드 크기를 조정할 수 있습니다.

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

import argparse
import logging
import sys

from deploy_requests import DeployRequests


def _parse_args():
    """ Parses the arguments provided on the command line when executing
this
        script and returns the resulting namespace. If all required
arguments
        are not provided, an error message indicating the mismatch is
printed and
        the script will exit.
    """

    parser = argparse.ArgumentParser(description=(
        'Uses the ONTAP Select Deploy API to resize the nodes in the
cluster.'
```

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


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

    cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args
```

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


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

    changes = {'admin_password': parsed_args.ontap_password}

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

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

    return changes


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

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

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

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

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

```
parsed_args.cluster)
        return 1

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

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