# Automatisieren mit REST

## ONTAP Select

NetApp
January 29, 2026

# Inhalt

# Automatisieren mit REST

# Konzepte

## REST-Webservices-Grundlage für die Bereitstellung und Verwaltung von ONTAP Select Clustern

Representational State Transfer (REST) ist ein Stil zur Erstellung verteilter Webanwendungen. Bei der Entwicklung einer Webservice-API etabliert es eine Reihe von Technologien und Best Practices für die Bereitstellung serverbasierter Ressourcen und die Verwaltung ihrer Zustände. REST nutzt gängige Protokolle und Standards und bietet so eine flexible Grundlage für die Bereitstellung und Verwaltung von ONTAP Select Clustern.

### Architektur und klassische Einschränkungen

REST wurde von Roy Fielding in seiner Doktorarbeit formal formuliert "Dissertation" an der UC Irvine im Jahr 2000. Es definiert einen Architekturstil durch eine Reihe von Einschränkungen, die zusammengenommen webbasierte Anwendungen und die zugrunde liegenden Protokolle verbessern. Die Einschränkungen etablieren eine RESTful-Webdienstanwendung basierend auf einer Client/Server-Architektur unter Verwendung eines zustandslosen Kommunikationsprotokolls.

### Ressourcen und staatliche Vertretung

Ressourcen sind die grundlegenden Komponenten eines webbasierten Systems. Zu den ersten Entwurfsaufgaben beim Erstellen einer REST-Webdienstanwendung gehören:

- Identifizierung system- oder serverbasierter Ressourcen. Jedes System nutzt und verwaltet Ressourcen. Eine Ressource kann eine Datei, eine Geschäftstransaktion, ein Prozess oder eine Verwaltungseinheit sein. Eine der ersten Aufgaben beim Entwurf einer Anwendung auf Basis von REST-Webdiensten ist die Identifizierung der Ressourcen.
- Definition von Ressourcenzuständen und zugehörigen Zustandsoperationen Ressourcen befinden sich immer in einem von einer begrenzten Anzahl von Zuständen. Die Zustände sowie die zugehörigen Operationen, die zur Zustandsänderung verwendet werden, müssen klar definiert sein.

Zwischen Client und Server werden Nachrichten ausgetauscht, um auf die Ressourcen zuzugreifen und ihren Status gemäß dem allgemeinen CRUD-Modell (Create, Read, Update und Delete) zu ändern.

### URI-Endpunkte

Jede REST-Ressource muss mithilfe eines klar definierten Adressierungsschemas definiert und bereitgestellt werden. Die Endpunkte, an denen die Ressourcen lokalisiert und identifiziert werden, verwenden einen Uniform Resource Identifier (URI). Der URI bietet einen allgemeinen Rahmen für die Erstellung eines eindeutigen Namens für jede Ressource im Netzwerk. Der Uniform Resource Locator (URL) ist ein URI-Typ, der in Webdiensten zur Identifizierung und zum Zugriff auf Ressourcen verwendet wird. Ressourcen werden typischerweise in einer hierarchischen Struktur ähnlich einem Dateiverzeichnis bereitgestellt.

### HTTP-Nachrichten

HTTP (Hypertext Transfer Protocol) ist das Protokoll, das vom Webdienst-Client und -Server zum Austausch von Anforderungs- und Antwortnachrichten zu Ressourcen verwendet wird. Beim Entwurf einer

Webdienstanwendung werden HTTP-Verben (wie GET und POST) den Ressourcen und den entsprechenden Statusverwaltungsaktionen zugeordnet.

HTTP ist zustandslos. Um eine Reihe zusammengehöriger Anfragen und Antworten einer Transaktion zuzuordnen, müssen daher zusätzliche Informationen in die HTTP-Header der Anfrage-/Antwort-Datenflüsse aufgenommen werden.

#### JSON-Formatierung

Informationen können auf verschiedene Weise strukturiert und zwischen Client und Server übertragen werden. Die beliebteste Option (und die mit der Deploy REST API verwendete) ist JavaScript Object Notation (JSON). JSON ist ein Industriestandard für die Darstellung einfacher Datenstrukturen in Klartext und wird zur Übertragung von Statusinformationen verwendet, die die Ressourcen beschreiben.

## So greifen Sie auf die ONTAP Select Deploy API zu

Aufgrund der inhärenten Flexibilität von REST-Webdiensten kann auf die ONTAP Select Deploy API auf verschiedene Arten zugegriffen werden.

#### Native Benutzeroberfläche des Bereitstellungsprogramms

Der primäre Zugriff auf die API erfolgt über die ONTAP Select Deploy-Webbenutzeroberfläche. Der Browser ruft die API auf und formatiert die Daten entsprechend dem Design der Benutzeroberfläche neu. Sie können auch über die Befehlszeilenschnittstelle des Deploy-Dienstprogramms auf die API zugreifen.

#### ONTAP Select Deploy Online-Dokumentationsseite

Die Online-Dokumentationsseite von ONTAP Select Deploy bietet einen alternativen Zugriffspunkt bei Verwendung eines Browsers. Neben der Möglichkeit, einzelne API-Aufrufe direkt auszuführen, enthält die Seite auch eine detaillierte Beschreibung der API, einschließlich Eingabeparametern und weiteren Optionen für jeden Aufruf. Die API-Aufrufe sind in verschiedene Funktionsbereiche oder Kategorien unterteilt.

#### Benutzerdefiniertes Programm

Sie können mit verschiedenen Programmiersprachen und Tools auf die Deploy-API zugreifen. Beliebte Optionen sind Python, Java und cURL. Ein Programm, Skript oder Tool, das die API nutzt, fungiert als REST-Webservice-Client. Die Verwendung einer Programmiersprache ermöglicht Ihnen ein besseres Verständnis der API und bietet die Möglichkeit, ONTAP Select Bereitstellungen zu automatisieren.

## ONTAP Select Deploy API-Versionierung

Die in ONTAP Select Deploy enthaltene REST-API erhält eine Versionsnummer. Die API-Versionsnummer ist unabhängig von der Deploy-Versionsnummer. Sie sollten sich über die in Ihrer Deploy-Version enthaltene API-Version und die möglichen Auswirkungen auf Ihre Nutzung der API im Klaren sein.

Die aktuelle Version des Deploy-Verwaltungsprogramms enthält Version 3 der REST-API. Frühere Versionen des Deploy-Dienstprogramms enthalten die folgenden API-Versionen:

**Bereitstellen von 2.8 und höher**

ONTAP Select Deploy 2.8 und alle späteren Versionen enthalten Version 3 der REST-API.

**Bereitstellen von 2.7.2 und früher**

ONTAP Select Deploy 2.7.2 und alle früheren Versionen enthalten Version 2 der REST-API.

> Die Versionen 2 und 3 der REST-API sind nicht kompatibel. Wenn Sie von einer früheren Version, die Version 2 der API enthält, auf Deploy 2.8 oder höher aktualisieren, müssen Sie den gesamten vorhandenen Code aktualisieren, der direkt auf die API zugreift, sowie alle Skripts, die die Befehlszeilenschnittstelle verwenden.

## Grundlegende Betriebsmerkmale der ONTAP Select Deploy API

REST bietet zwar einen gemeinsamen Satz an Technologien und Best Practices, die Details der einzelnen APIs können jedoch je nach Designentscheidung variieren. Machen Sie sich vor der Verwendung der ONTAP Select Deploy API mit den Details und Betriebsmerkmalen vertraut.

### Hypervisor-Host im Vergleich zu ONTAP Select Knoten

Ein Hypervisor-Host ist die zentrale Hardwareplattform, auf der eine virtuelle ONTAP Select Maschine gehostet wird. Wenn eine virtuelle ONTAP Select Maschine auf einem Hypervisor-Host bereitgestellt und aktiv ist, gilt sie als ONTAP Select-Knoten. Mit Version 3 der Deploy REST API sind Host- und Knotenobjekte getrennt und unterschiedlich. Dies ermöglicht eine Eins-zu-viele-Beziehung, bei der ein oder mehrere ONTAP Select Knoten auf demselben Hypervisor-Host ausgeführt werden können.

### Objektkennungen

Jede Ressourceninstanz oder jedes Objekt erhält bei der Erstellung eine eindeutige Kennung. Diese Kennungen sind innerhalb einer bestimmten Instanz von ONTAP Select Deploy global eindeutig. Nach dem Ausführen eines API-Aufrufs, der eine neue Objektinstanz erstellt, wird der zugehörige ID-Wert im folgenden Format an den Aufrufer zurückgegeben: `location` Header der HTTP-Antwort. Sie können die Kennung extrahieren und sie bei nachfolgenden Aufrufen verwenden, wenn Sie auf die Ressourceninstanz verweisen.

> Inhalt und interne Struktur der Objektkennungen können sich jederzeit ändern. Sie sollten die Kennungen bei den entsprechenden API-Aufrufen nur nach Bedarf verwenden, wenn Sie auf die zugehörigen Objekte verweisen.

### Anforderungskennungen

Jeder erfolgreichen API-Anfrage wird eine eindeutige Kennung zugewiesen. Die Kennung wird in der `request-id` Header der zugehörigen HTTP-Antwort. Mit einer Anforderungskennung können Sie die Aktivitäten einer einzelnen API-Anforderungs-Antwort-Transaktion kollektiv referenzieren. Beispielsweise können Sie alle Ereignismeldungen für eine Transaktion anhand der Anforderungs-ID abrufen.

### Synchrone und asynchrone Aufrufe

Es gibt zwei Hauptmethoden, mit denen ein Server eine von einem Client empfangene HTTP-Anforderung ausführt:

- Synchron Der Server führt die Anfrage sofort aus und antwortet mit einem Statuscode von 200, 201 oder 204.
- Asynchron: Der Server akzeptiert die Anfrage und antwortet mit dem Statuscode 202. Dies zeigt an, dass der Server die Clientanfrage akzeptiert und eine Hintergrundaufgabe zum Abschließen der Anfrage

gestartet hat. Der endgültige Erfolg oder Misserfolg ist nicht sofort verfügbar und muss durch zusätzliche API-Aufrufe ermittelt werden.

#### Bestätigen Sie den Abschluss eines lang laufenden Auftrags

Im Allgemeinen werden Vorgänge, deren Ausführung längere Zeit in Anspruch nehmen kann, asynchron mithilfe einer Hintergrundaufgabe auf dem Server verarbeitet. Mit der Deploy-REST-API wird jede Hintergrundaufgabe durch ein Job-Objekt verankert, das die Aufgabe verfolgt und Informationen wie den aktuellen Status bereitstellt. Ein Job-Objekt mit seiner eindeutigen Kennung wird in der HTTP-Antwort zurückgegeben, nachdem eine Hintergrundaufgabe erstellt wurde.

Sie können das Job-Objekt direkt abfragen, um den Erfolg oder Misserfolg des zugehörigen API-Aufrufs zu ermitteln. Weitere Informationen finden Sie unter *Asynchrone Verarbeitung mit dem Job-Objekt*.

Neben der Verwendung des Job-Objekts gibt es noch weitere Möglichkeiten, den Erfolg oder Misserfolg einer Anfrage zu bestimmen, darunter:

- Ereignismeldungen: Sie können alle Ereignismeldungen, die einem bestimmten API-Aufruf zugeordnet sind, mithilfe der Anforderungs-ID abrufen, die mit der ursprünglichen Antwort zurückgegeben wurde. Die Ereignismeldungen enthalten in der Regel einen Hinweis auf Erfolg oder Misserfolg und können auch beim Debuggen eines Fehlerzustands hilfreich sein.
- Ressourcenzustand oder -status Mehrere der Ressourcen verfügen über einen Zustands- oder Statuswert, den Sie abfragen können, um indirekt den Erfolg oder Misserfolg einer Anfrage zu bestimmen.

#### Sicherheit

Die Deploy-API verwendet die folgenden Sicherheitstechnologien:

- Transport Layer Security Der gesamte zwischen Deploy-Server und Client gesendete Netzwerkverkehr wird über TLS verschlüsselt. Die Verwendung des HTTP-Protokolls über einen unverschlüsselten Kanal wird nicht unterstützt. TLS Version 1.2 wird unterstützt.
- HTTP-Authentifizierung Für jede API-Transaktion wird die Basisauthentifizierung verwendet. Jeder Anfrage wird ein HTTP-Header hinzugefügt, der den Benutzernamen und das Kennwort in einer Base64-Zeichenfolge enthält.

## Anforderungs- und Antwort-API-Transaktion für ONTAP Select

Jeder Deploy-API-Aufruf wird als HTTP-Anfrage an die Deploy-VM ausgeführt, die eine entsprechende Antwort an den Client generiert. Dieses Anfrage-/Antwort-Paar gilt als API-Transaktion. Bevor Sie die Deploy-API verwenden, sollten Sie mit den verfügbaren Eingabevariablen zur Steuerung einer Anfrage und dem Inhalt der Antwortausgabe vertraut sein.

#### Eingabevariablen, die eine API-Anfrage steuern

Sie können die Verarbeitung eines API-Aufrufs über in der HTTP-Anforderung festgelegte Parameter steuern.

##### Anforderungsheader

Sie müssen mehrere Header in die HTTP-Anforderung aufnehmen, darunter:

- Inhaltstyp: Wenn der Anforderungstext JSON enthält, muss dieser Header auf application/json gesetzt werden.

- Akzeptieren. Wenn der Antworttext JSON enthält, muss dieser Header auf application/json gesetzt werden.
- Autorisierung: Die Basisauthentifizierung muss mit dem Benutzernamen und dem Kennwort in einer Base64-Zeichenfolge codiert sein.

#### Anforderungstext

Der Inhalt des Anforderungstexts variiert je nach Aufruf. Der HTTP-Anforderungstext besteht aus einem der folgenden Elemente:

- JSON-Objekt mit Eingabevariablen (z. B. dem Namen eines neuen Clusters)
- Leer

#### Filterobjekte

Wenn Sie einen API-Aufruf mit GET ausführen, können Sie die zurückgegebenen Objekte basierend auf einem beliebigen Attribut einschränken oder filtern. Sie können beispielsweise einen genauen Wert angeben, der übereinstimmen soll:

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

Neben der exakten Übereinstimmung stehen weitere Operatoren zur Verfügung, um eine Reihe von Objekten über einen Wertebereich zurückzugeben. ONTAP Select unterstützt die unten gezeigten Filteroperatoren.

| Operator | Beschreibung |
|---|---|
| = | Gleich |
| < | Weniger als |
| > | Größer als |
| ⇐ | Kleiner oder gleich |
| >= | Größer als oder gleich |
| | Oder |
| ! | Ungleich |
| * | Gieriger Platzhalter |

Sie können auch eine Reihe von Objekten zurückgeben, basierend darauf, ob ein bestimmtes Feld festgelegt ist oder nicht, indem Sie das Schlüsselwort „Null“ oder seine Negation (!null) als Teil der Abfrage verwenden.

#### Auswählen von Objektfeldern

Standardmäßig gibt ein API-Aufruf mit GET nur die Attribute zurück, die das Objekt bzw. die Objekte eindeutig identifizieren. Dieser Mindestsatz an Feldern dient als Schlüssel für jedes Objekt und variiert je nach Objekttyp. Sie können zusätzliche Objekteigenschaften mit dem Abfrageparameter „Felder“ wie folgt auswählen:

- Preiswerte Felder angeben `fields=*` um die Objektfelder abzurufen, die im lokalen Serverspeicher verwaltet werden oder für deren Zugriff nur eine geringe Verarbeitung erforderlich ist.
- Teure Felder angeben `fields=**` um alle Objektfelder abzurufen, einschließlich derjenigen, für deren Zugriff zusätzliche Serververarbeitung erforderlich ist.
- Benutzerdefinierte Feldauswahl Verwenden `fields=FIELDNAME` um das gewünschte Feld genau anzugeben. Wenn Sie mehrere Felder anfordern, müssen die Werte durch Kommas und ohne Leerzeichen

getrennt werden.

Als Best Practice sollten Sie immer die gewünschten Felder angeben. Rufen Sie die kostengünstigen und teuren Felder nur bei Bedarf ab. Die Klassifizierung in kostengünstige und teure Felder wird von NetApp anhand interner Performanceanalysen festgelegt. Die Klassifizierung für ein bestimmtes Feld kann sich jederzeit ändern.

#### Sortieren von Objekten im Ausgabesatz

Die Datensätze einer Ressourcensammlung werden in der vom Objekt definierten Standardreihenfolge zurückgegeben. Sie können die Reihenfolge mithilfe des Abfrageparameters order_by mit dem Feldnamen und der Sortierrichtung wie folgt ändern:
`order_by=<field name> asc|desc`

Sie können beispielsweise das Typfeld in absteigender Reihenfolge und anschließend die ID in aufsteigender Reihenfolge sortieren:
`order_by=type desc, id asc`

Wenn Sie mehrere Parameter angeben, müssen Sie die Felder durch ein Komma trennen.

#### Pagination

Wenn Sie einen API-Aufruf mit GET ausführen, um auf eine Sammlung von Objekten desselben Typs zuzugreifen, werden standardmäßig alle übereinstimmenden Objekte zurückgegeben. Bei Bedarf können Sie die Anzahl der zurückgegebenen Datensätze mit dem Abfrageparameter max_records in der Anfrage begrenzen. Beispiel:
`max_records=20`

Bei Bedarf können Sie diesen Parameter mit anderen Abfrageparametern kombinieren, um das Ergebnis einzugrenzen. Beispielsweise gibt die folgende Abfrage bis zu 10 Systemereignisse zurück, die nach der angegebenen Zeit generiert wurden:
`time⇒ 2019-04-04T15:41:29.140265Z&max_records=10`

Sie können mehrere Anfragen stellen, um die Ereignisse (oder einen beliebigen Objekttyp) durchzublättern. Jeder nachfolgende API-Aufruf sollte einen neuen Zeitwert basierend auf dem letzten Ereignis im letzten Ergebnissatz verwenden.

### Interpretieren einer API-Antwort

Jede API-Anfrage generiert eine Antwort an den Client. Sie können die Antwort überprüfen, um festzustellen, ob sie erfolgreich war, und bei Bedarf weitere Daten abrufen.

#### HTTP-Statuscode

Die von der Deploy REST API verwendeten HTTP-Statuscodes werden unten beschrieben.

| Code | Bedeutung | Beschreibung |
|---|---|---|
| 200 | OK | Zeigt den Erfolg von Aufrufen an, die kein neues Objekt erstellen. |
| 201 | Erstellt | Ein Objekt wurde erfolgreich erstellt. Der Location-Antwortheader enthält die eindeutige Kennung für das Objekt. |

| Code | Bedeutung | Beschreibung |
|---|---|---|
| 202 | Akzeptiert | Zur Ausführung der Anforderung wurde ein zeitintensiver Hintergrundjob gestartet, der Vorgang ist jedoch noch nicht abgeschlossen. |
| 400 | Ungültige Anforderung | Die Anfrageeingabe wird nicht erkannt oder ist unpassend. |
| 403 | Verboten | Der Zugriff wird aufgrund eines Autorisierungsfehlers verweigert. |
| 404 | Nicht gefunden | Die in der Anfrage genannte Ressource existiert nicht. |
| 405 | Methode nicht zulässig | Das HTTP-Verb in der Anforderung wird für die Ressource nicht unterstützt. |
| 409 | Konflikt | Der Versuch, ein Objekt zu erstellen, ist fehlgeschlagen, da das Objekt bereits vorhanden ist. |
| 500 | Interner Fehler | Auf dem Server ist ein allgemeiner interner Fehler aufgetreten. |
| 501 | Nicht implementiert | Die URI ist bekannt, kann die Anfrage jedoch nicht ausführen. |

#### Antwortheader

Die vom Deploy-Server generierte HTTP-Antwort enthält mehrere Header, darunter:

- Anfrage-ID: Jeder erfolgreichen API-Anfrage wird eine eindeutige Anfragekennung zugewiesen.
- Standort: Wenn ein Objekt erstellt wird, enthält der Standort-Header die vollständige URL zum neuen Objekt einschließlich der eindeutigen Objektkennung.

#### Antworttext

Der Inhalt der mit einer API-Anfrage verknüpften Antwort unterscheidet sich je nach Objekt, Verarbeitungstyp und Erfolg oder Misserfolg der Anfrage. Der Antworttext wird in JSON gerendert.

- Einzelnes Objekt Ein einzelnes Objekt kann basierend auf der Anforderung mit einer Reihe von Feldern zurückgegeben werden. Beispielsweise können Sie GET verwenden, um ausgewählte Eigenschaften eines Clusters mithilfe der eindeutigen Kennung abzurufen.
- Mehrere Objekte Mehrere Objekte aus einer Ressourcensammlung können zurückgegeben werden. In allen Fällen wird ein einheitliches Format verwendet, mit `num_records` Gibt die Anzahl der Datensätze und Datensätze an, die ein Array der Objektinstanzen enthalten. Sie können beispielsweise alle in einem bestimmten Cluster definierten Knoten abrufen.
- Job-Objekt: Wenn ein API-Aufruf asynchron verarbeitet wird, wird ein Job-Objekt zurückgegeben, das die Hintergrundaufgabe verankert. Beispielsweise wird die POST-Anforderung zum Bereitstellen eines Clusters asynchron verarbeitet und gibt ein Job-Objekt zurück.
- Fehlerobjekt: Wenn ein Fehler auftritt, wird immer ein Fehlerobjekt zurückgegeben. Beispielsweise erhalten Sie eine Fehlermeldung, wenn Sie versuchen, einen Cluster mit einem bereits vorhandenen Namen zu erstellen.
- Leer: In bestimmten Fällen werden keine Daten zurückgegeben und der Antworttext ist leer. Beispielsweise ist der Antworttext leer, nachdem mit DELETE ein vorhandener Host gelöscht wurde.

## Asynchrone Verarbeitung mit dem Job-Objekt für ONTAP Select

Einige Deploy-API-Aufrufe, insbesondere solche zum Erstellen oder Ändern einer Ressource, können länger dauern als andere Aufrufe. ONTAP Select Deploy verarbeitet diese lang laufenden Anfragen asynchron.

### Asynchrone Anforderungen, beschrieben mithilfe des Job-Objekts

Nach einem asynchron ausgeführten API-Aufruf zeigt der HTTP-Antwortcode 202 an, dass die Anfrage erfolgreich validiert und akzeptiert, aber noch nicht abgeschlossen wurde. Die Anfrage wird als Hintergrundaufgabe verarbeitet, die nach der ersten HTTP-Antwort an den Client weiterläuft. Die Antwort enthält das Job-Objekt, das die Anfrage verankert, einschließlich seiner eindeutigen Kennung.

Um zu ermitteln, welche API-Aufrufe asynchron ausgeführt werden, sollten Sie auf der Online-Dokumentationsseite von ONTAP Select Deploy nachsehen.

### Abfragen des mit einer API-Anforderung verknüpften Job-Objekts

Das in der HTTP-Antwort zurückgegebene Job-Objekt enthält mehrere Eigenschaften. Sie können die Statuseigenschaft abfragen, um festzustellen, ob die Anforderung erfolgreich abgeschlossen wurde. Ein Job-Objekt kann einen der folgenden Zustände aufweisen:

- In der Warteschlange
- Wird ausgeführt
- Erfolg
- Versagen

Es gibt zwei Techniken, die Sie beim Abfragen eines Job-Objekts verwenden können, um einen Endzustand für die Aufgabe zu erkennen (entweder Erfolg oder Misserfolg):

- Standard-Polling-Anfrage Aktueller Jobstatus wird sofort zurückgegeben
- Der Auftragsstatus für lange Polling-Anfragen wird nur zurückgegeben, wenn eines der folgenden Ereignisse eintritt:
  - Der Status hat sich vor dem in der Abfrageanforderung angegebenen Datums-/Uhrzeitwert geändert.
  - Timeout-Wert ist abgelaufen (1 bis 120 Sekunden)

Standard-Polling und Long-Polling verwenden denselben API-Aufruf zum Abfragen eines Job-Objekts. Eine Long-Polling-Anforderung enthält jedoch zwei Abfrageparameter: `poll_timeout` Und `last_modified`.

Sie sollten immer Long Polling verwenden, um die Arbeitslast auf der virtuellen Bereitstellungsmaschine zu reduzieren.

### Allgemeine Vorgehensweise zum Ausgeben einer asynchronen Anfrage

Sie können das folgende allgemeine Verfahren verwenden, um einen asynchronen API-Aufruf abzuschließen:

1. Führen Sie den asynchronen API-Aufruf aus.
2. Erhalten Sie eine HTTP-Antwort 202, die die erfolgreiche Annahme der Anfrage anzeigt.
3. Extrahieren Sie die Kennung für das Job-Objekt aus dem Antworttext.

4. Führen Sie innerhalb einer Schleife in jedem Zyklus Folgendes aus:
   a. Holen Sie sich den aktuellen Status des Jobs mit einer Long-Poll-Anfrage
   b. Wenn sich der Job in einem nicht-terminalen Zustand befindet (in der Warteschlange, läuft), führen Sie die Schleife erneut aus.
5. Stoppen Sie, wenn der Job einen Endzustand (Erfolg, Fehler) erreicht.

# Zugriff mit einem Browser

## Bevor Sie mit einem Browser auf die ONTAP Select Deploy API zugreifen

Bevor Sie die Online-Dokumentationsseite „Deploy“ verwenden, sollten Sie sich über mehrere Dinge im Klaren sein.

#### Bereitstellungsplan

Wenn Sie API-Aufrufe im Rahmen bestimmter Bereitstellungs- oder Verwaltungsaufgaben ausführen möchten, sollten Sie einen Bereitstellungsplan erstellen. Diese Pläne können formell oder informell sein und enthalten in der Regel Ihre Ziele und die zu verwendenden API-Aufrufe. Weitere Informationen finden Sie unter Workflow-Prozesse mit der Deploy REST API.

#### JSON-Beispiele und Parameterdefinitionen

Jeder API-Aufruf wird auf der Dokumentationsseite in einem einheitlichen Format beschrieben. Der Inhalt umfasst Implementierungshinweise, Abfrageparameter und HTTP-Statuscodes. Darüber hinaus können Sie Details zum JSON-Code anzeigen, der mit den API-Anfragen und -Antworten verwendet wird:

- Beispielwert: Wenn Sie bei einem API-Aufruf auf *Beispielwert* klicken, wird eine typische JSON-Struktur für den Aufruf angezeigt. Sie können das Beispiel nach Bedarf anpassen und als Eingabe für Ihre Anfrage verwenden.
- Modell Wenn Sie auf *Modell* klicken, wird eine vollständige Liste der JSON-Parameter mit einer Beschreibung für jeden Parameter angezeigt.

#### Vorsicht beim Ausgeben von API-Aufrufen

Alle API-Vorgänge, die Sie über die Dokumentationsseite „Bereitstellen“ ausführen, sind Live-Vorgänge. Achten Sie darauf, nicht versehentlich Konfigurationen oder andere Daten zu erstellen, zu aktualisieren oder zu löschen.

## Greifen Sie auf die Dokumentationsseite zu ONTAP Select Deploy zu

Sie müssen auf die Online-Dokumentationsseite von ONTAP Select Deploy zugreifen, um die API-Dokumentation anzuzeigen und manuell einen API-Aufruf zu tätigen.

#### Bevor Sie beginnen

Sie müssen über Folgendes verfügen:

- IP-Adresse oder Domänenname der ONTAP Select Deploy-Virtual-Machine
- Benutzername und Passwort für den Administrator

#### Schritte

1. Geben Sie die URL in Ihren Browser ein und drücken Sie **Enter**:

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

2. Sign in .

**Ergebnis**

Die Webseite zur Bereitstellungsdokumentation wird mit den nach Kategorien sortierten Anrufen unten auf der Seite angezeigt.

## Verstehen und Ausführen eines ONTAP Select Deploy API-Aufrufs

Die Details aller API-Aufrufe werden auf der Online-Dokumentationswebseite von ONTAP Select Deploy in einem einheitlichen Format dokumentiert und angezeigt. Durch das Verständnis eines einzelnen API-Aufrufs können Sie auf die Details aller API-Aufrufe zugreifen und diese interpretieren.

**Bevor Sie beginnen**

Sie müssen bei der Online-Dokumentationswebseite von ONTAP Select Deploy angemeldet sein. Sie benötigen die eindeutige Kennung, die Ihrem ONTAP Select Cluster bei der Erstellung des Clusters zugewiesen wurde.

**Informationen zu diesem Vorgang**

Sie können die Konfigurationsinformationen eines ONTAP Select Clusters anhand seiner eindeutigen Kennung abrufen. In diesem Beispiel werden alle als kostengünstig klassifizierten Felder zurückgegeben. Es empfiehlt sich jedoch, nur die benötigten Felder anzufordern.

**Schritte**

1. Scrollen Sie auf der Hauptseite nach unten und klicken Sie auf **Cluster**.
2. Klicken Sie auf **GET /clusters/{cluster_id}**, um die Details des API-Aufrufs anzuzeigen, der zum Zurückgeben von Informationen zu einem ONTAP Select Cluster verwendet wird.

# Workflow-Prozesse

## Bevor Sie die ONTAP Select Deploy API-Workflows verwenden

Sie sollten sich darauf vorbereiten, die Workflow-Prozesse zu überprüfen und zu verwenden.

**Verstehen Sie die in den Workflows verwendeten API-Aufrufe**

Die Online-Dokumentationsseite von ONTAP Select enthält die Details jedes REST-API-Aufrufs. Anstatt diese Details hier zu wiederholen, enthält jeder in den Workflow-Beispielen verwendete API-Aufruf nur die Informationen, die Sie zum Auffinden des Aufrufs auf der Dokumentationsseite benötigen. Nachdem Sie einen bestimmten API-Aufruf gefunden haben, können Sie die vollständigen Details des Aufrufs überprüfen, einschließlich der Eingabeparameter, Ausgabeformate, HTTP-Statuscodes und des Anfrageverarbeitungstyps.

Die folgenden Informationen sind für jeden API-Aufruf innerhalb eines Workflows enthalten, um das Auffinden des Aufrufs auf der Dokumentationsseite zu erleichtern:

- Kategorie Die API-Aufrufe sind auf der Dokumentationsseite in funktional verwandte Bereiche oder Kategorien unterteilt. Um einen bestimmten API-Aufruf zu finden, scrollen Sie zum Ende der Seite und klicken Sie auf die entsprechende API-Kategorie.
- HTTP-Verb: Das HTTP-Verb identifiziert die für eine Ressource ausgeführte Aktion. Jeder API-Aufruf wird über ein einzelnes HTTP-Verb ausgeführt.
- Pfad: Der Pfad bestimmt die spezifische Ressource, auf die die Aktion im Rahmen eines Aufrufs angewendet wird. Die Pfadzeichenfolge wird an die Kern-URL angehängt, um die vollständige URL zur Identifizierung der Ressource zu bilden.

#### Erstellen Sie eine URL für den direkten Zugriff auf die REST-API

Zusätzlich zur ONTAP Select Dokumentationsseite können Sie auch direkt über eine Programmiersprache wie Python auf die Deploy REST API zugreifen. In diesem Fall unterscheidet sich die Kern-URL geringfügig von der URL, die beim Zugriff auf die Online-Dokumentationsseite verwendet wird. Beim direkten Zugriff auf die API müssen Sie /api an die Domänen- und Portzeichenfolge anhängen. Beispiel: `http://deploy.mycompany.com/api`

## Workflow 1: Erstellen Sie einen ONTAP Select Single-Node-Evaluierungscluster auf ESXi

Sie können einen ONTAP Select Cluster mit einem Knoten auf einem von vCenter verwalteten VMware ESXi-Host bereitstellen. Der Cluster wird mit einer Evaluierungslizenz erstellt.

Der Workflow zur Clustererstellung unterscheidet sich in den folgenden Situationen:

- Der ESXi-Host wird nicht von vCenter verwaltet (eigenständiger Host).
- Innerhalb des Clusters werden mehrere Knoten oder Hosts verwendet
- Der Cluster wird in einer Produktionsumgebung mit einer erworbenen Lizenz bereitgestellt
- Der KVM-Hypervisor wird anstelle von VMware ESXi verwendet

#### 1. Registrieren Sie die Anmeldeinformationen für den vCenter-Server

Bei der Bereitstellung auf einem ESXi-Host, der von einem vCenter-Server verwaltet wird, müssen Sie vor der Registrierung des Hosts Anmeldeinformationen hinzufügen. Das Deploy-Verwaltungsprogramm kann diese Anmeldeinformationen dann zur Authentifizierung bei vCenter verwenden.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Einsetzen | POST | /Sicherheit/Anmeldeinformationen |

**Locken**

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

**JSON-Eingabe (Schritt 01)**

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

**Verarbeitungstyp**

Asynchron

**Ausgabe**

- Anmeldeinformations-ID im Standortantwortheader
- Job-Objekt

### 2. Registrieren Sie einen Hypervisor-Host

Sie müssen einen Hypervisor-Host hinzufügen, auf dem die virtuelle Maschine mit dem ONTAP Select Knoten ausgeführt wird.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | POST | /Gastgeber |

**Locken**

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

**JSON-Eingabe (Schritt 02)**

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

**Verarbeitungstyp**

Asynchron

**Ausgabe**

- Host-ID im Standortantwortheader
- Job-Objekt

### 3. Erstellen Sie einen Cluster

Wenn Sie einen ONTAP Select Cluster erstellen, wird die grundlegende Clusterkonfiguration registriert und die Knotennamen werden automatisch von Deploy generiert.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | POST | /Cluster |

**Locken**

Der Abfrageparameter node_count sollte für einen Einzelknotencluster auf 1 gesetzt werden.

```
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-Eingabe (Schritt 03)**

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

- Cluster-ID im Standortantwortheader

### 4. Konfigurieren Sie den Cluster

Es gibt mehrere Attribute, die Sie im Rahmen der Clusterkonfiguration angeben müssen.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | PATCH | /clusters/{cluster_id} |

**Locken**

Sie müssen die Cluster-ID angeben.

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

**JSON-Eingabe (Schritt 04)**

```
{
  "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"}
}
```

**Verarbeitungstyp**

Synchron

**Ausgabe**

Keine

#### 5. Rufen Sie den Knotennamen ab

Das Verwaltungsdienstprogramm „Deploy“ generiert beim Erstellen eines Clusters automatisch die Knotenkennungen und -namen. Bevor Sie einen Knoten konfigurieren können, müssen Sie die zugewiesene ID abrufen.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | ERHALTEN | /clusters/{cluster_id}/nodes |

**Locken**

Sie müssen die Cluster-ID angeben.

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

- Array-Datensätze, die jeweils einen einzelnen Knoten mit der eindeutigen ID und dem Namen beschreiben

#### 6. Konfigurieren Sie die Knoten

Sie müssen die Basiskonfiguration für den Knoten angeben. Dies ist der erste von drei API-Aufrufen, die zum Konfigurieren eines Knotens verwendet werden.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | WEG | /clusters/{cluster_id}/nodes/{node_id} |

**Locken**

Sie müssen die Cluster-ID und die Knoten-ID angeben.

```
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-Eingabe (Schritt 06)**

Sie müssen die Host-ID angeben, auf der der ONTAP Select Knoten ausgeführt wird.

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

Keine

### 7. Abrufen der Knotennetzwerke

Sie müssen die vom Knoten im Einzelknotencluster verwendeten Daten- und Verwaltungsnetzwerke identifizieren. Das interne Netzwerk wird bei einem Einzelknotencluster nicht verwendet.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | ERHALTEN | /clusters/{cluster_id}/nodes/{node_id}/networks |

**Locken**

Sie müssen die Cluster-ID und die Knoten-ID angeben.

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

- Array aus zwei Datensätzen, die jeweils ein einzelnes Netzwerk für den Knoten beschreiben, einschließlich der eindeutigen ID und des Zwecks

### 8. Konfigurieren Sie die Knotenvernetzung

Sie müssen die Daten- und Verwaltungsnetzwerke konfigurieren. Das interne Netzwerk wird bei einem Einzelknotencluster nicht verwendet.

Führen Sie den folgenden API-Aufruf zweimal aus, einmal für jedes Netzwerk.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | PATCH | /clusters/{cluster_id}/nodes/{node_id}/networks/{network_id} |

**Locken**

Sie müssen die Cluster-ID, Knoten-ID und Netzwerk-ID angeben.

```
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-Eingabe (Schritt 08)**

Sie müssen den Namen des Netzwerks angeben.

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

Keine

### 9. Konfigurieren Sie den Knotenspeicherpool

Der letzte Schritt bei der Konfiguration eines Knotens besteht im Anschließen eines Speicherpools. Sie können die verfügbaren Speicherpools über den vSphere-Webclient oder optional über die Deploy-REST-API ermitteln.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | PATCH | /clusters/{cluster_id}/nodes/{node_id}/networks/{network_id} |

**Locken**

Sie müssen die Cluster-ID, Knoten-ID und Netzwerk-ID angeben.

```
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-Eingabe (Schritt 09)**

Die Poolkapazität beträgt 2 TB.

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

**Verarbeitungstyp**

Synchron

**Ausgabe**

Keine

### 10. Bereitstellen des Clusters

Nachdem Cluster und Knoten konfiguriert wurden, können Sie den Cluster bereitstellen.

| Kategorie | HTTP-Verb | Weg |
|---|---|---|
| Cluster | POST | /clusters/{cluster_id}/deploy |

**Locken**

Sie müssen die Cluster-ID angeben.

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

**JSON-Eingabe (Schritt 10)**

Sie müssen das Kennwort für das ONTAP Administratorkonto angeben.

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

**Verarbeitungstyp**

Asynchron

**Ausgabe**

- Job-Objekt

**Ähnliche Informationen**

"Stellen Sie eine 90-tägige Testinstanz eines ONTAP Select Clusters bereit"

# Zugriff mit Python

## Bevor Sie auf die ONTAP Select Deploy API mit Python zugreifen

Sie müssen die Umgebung vorbereiten, bevor Sie die Python-Beispielskripts ausführen.

Bevor Sie die Python-Skripte ausführen, müssen Sie sicherstellen, dass die Umgebung richtig konfiguriert ist:

- Die neueste Version von Python2 muss installiert sein. Die Beispielcodes wurden mit Python2 getestet. Sie sollten auch auf Python3 portierbar sein, wurden jedoch nicht auf Kompatibilität getestet.
- Die Bibliotheken Requests und urllib3 müssen installiert sein. Sie können je nach Umgebung pip oder ein anderes Python-Verwaltungstool verwenden.
- Die Client-Workstation, auf der die Skripte ausgeführt werden, muss über Netzwerkzugriff auf die virtuelle ONTAP Select Deploy-Maschine verfügen.

Darüber hinaus müssen Sie über folgende Informationen verfügen:

- IP-Adresse der virtuellen Deploy-Maschine
- Benutzername und Kennwort eines Deploy-Administratorkontos

## Verstehen Sie die Python-Skripte für ONTAP Select Deploy

Mit den Python-Beispielskripten können Sie verschiedene Aufgaben ausführen. Sie sollten die Skripte verstehen, bevor Sie sie in einer Live-Deploy-Instanz verwenden.

**Gemeinsame Designmerkmale**

Die Skripte wurden mit den folgenden gemeinsamen Merkmalen entwickelt:

- Ausführen über die Befehlszeilenschnittstelle auf einem Client-Computer. Sie können die Python-Skripte von jedem entsprechend konfigurierten Client-Computer aus ausführen. Weitere Informationen finden Sie unter „Bevor Sie beginnen“.
- CLI-Eingabeparameter akzeptieren Jedes Skript wird an der CLI über Eingabeparameter gesteuert.
- Eingabedatei lesen Jedes Skript liest eine Eingabedatei entsprechend seinem Zweck. Beim Erstellen oder Löschen eines Clusters müssen Sie eine JSON-Konfigurationsdatei angeben. Beim Hinzufügen einer Knotenlizenz müssen Sie eine gültige Lizenzdatei angeben.
- Verwenden Sie ein gemeinsames Supportmodul. Das gemeinsame Supportmodul *deploy_requests.py* enthält eine einzelne Klasse. Es wird von jedem der Skripts importiert und verwendet.

#### Erstellen eines Clusters

Sie können einen ONTAP Select Cluster mit dem Skript cluster.py erstellen. Basierend auf den CLI-Parametern und dem Inhalt der JSON-Eingabedatei können Sie das Skript wie folgt an Ihre Bereitstellungsumgebung anpassen:

- Hypervisor: Sie können die Bereitstellung auf ESXI oder KVM durchführen (je nach Deploy-Version). Bei der Bereitstellung auf ESXi kann der Hypervisor von vCenter verwaltet werden oder ein eigenständiger Host sein.
- Clustergröße Sie können einen Cluster mit einem oder mehreren Knoten bereitstellen.
- Evaluierungs- oder Produktionslizenz Sie können einen Cluster mit einer Evaluierungs- oder erworbenen Lizenz für die Produktion bereitstellen.

Die CLI-Eingabeparameter für das Skript umfassen:

- Hostname oder IP-Adresse des Deploy-Servers
- Passwort für das Admin-Benutzerkonto
- Name der JSON-Konfigurationsdatei
- Verbose-Flag für die Nachrichtenausgabe

#### Hinzufügen einer Knotenlizenz

Wenn Sie einen Produktionscluster bereitstellen, müssen Sie mit dem Skript *add_license.py* für jeden Knoten eine Lizenz hinzufügen. Sie können die Lizenz vor oder nach der Bereitstellung des Clusters hinzufügen.

Die CLI-Eingabeparameter für das Skript umfassen:

- Hostname oder IP-Adresse des Deploy-Servers
- Passwort für das Admin-Benutzerkonto
- Name der Lizenzdatei
- ONTAP -Benutzername mit Berechtigungen zum Hinzufügen der Lizenz
- Passwort für den ONTAP -Benutzer

#### Löschen eines Clusters

Sie können einen vorhandenen ONTAP Select Cluster mit dem Skript *delete_cluster.py* löschen.

Die CLI-Eingabeparameter für das Skript umfassen:

- Hostname oder IP-Adresse des Deploy-Servers
- Passwort für das Admin-Benutzerkonto
- Name der JSON-Konfigurationsdatei

# Python-Codebeispiele

## Skript zum Erstellen eines ONTAP Select Clusters

Sie können das folgende Skript verwenden, um einen Cluster basierend auf im Skript definierten Parametern und einer JSON-Eingabedatei zu erstellen.

```
#!/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)
```

### JSON für Skript zum Erstellen eines ONTAP Select Clusters

Beim Erstellen oder Löschen eines ONTAP Select Clusters mithilfe der Python-Codebeispiele müssen Sie eine JSON-Datei als Eingabe für das Skript bereitstellen. Sie

können das entsprechende JSON-Beispiel basierend auf Ihren Bereitstellungsplänen kopieren und ändern.

**Einzelknotencluster auf ESXi**

```
{
  "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
          }
        ]
      }
    }
  ]
}
```

**Einzelknotencluster auf ESXi mit vCenter**

```
{
  "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
          }
```

```
        ]
      }
    }
  ]
}
```

### Einzelknotencluster auf KVM

```
{
  "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
          }
        ]
      }
    }
  ]
}
```

### Skript zum Hinzufügen einer ONTAP Select Knotenlizenz

Mit dem folgenden Skript können Sie eine Lizenz für einen ONTAP Select Knoten hinzufügen.

```
#!/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)
```

### Skript zum Löschen eines ONTAP Select Clusters

Sie können das folgende CLI-Skript verwenden, um einen vorhandenen Cluster zu löschen.

```
#!/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)
```

### Gemeinsames Support-Python-Modul für ONTAP Select

Alle Python-Skripte verwenden eine gemeinsame Python-Klasse in einem einzigen Modul.

```
#!/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}
```

### Skript zum Ändern der Größe von ONTAP Select Clusterknoten

Mit dem folgenden Skript können Sie die Größe der Knoten in einem ONTAP Select Cluster ändern.

```
#!/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())
```