Automatisieren mit REST

ONTAP Select

NetApp
November 08, 2026

This PDF was generated from https://docs.netapp.com/de-de/ontap-select-9191/concept_api_rest.html on November 08, 2026. Always check docs.netapp.com for the latest.

Inhalt

Automatisieren mit REST . . . 1
- Konzepte . . . 1
  - Informationen zu REST-Webdiensten für die Bereitstellung und Verwaltung von ONTAP Select Clustern . . . 1
  - Informationen zum Zugriff auf die ONTAP Select Deploy API . . . 2
  - Grundlegende Betriebsmerkmale der ONTAP Select Deploy API . . . 2
  - Anfrage- und Antwort-API-Transaktion für ONTAP Select . . . 4
  - Asynchrone Verarbeitung mit dem Job-Objekt für ONTAP Select . . . 7
- Zugriff mit einem Browser . . . 8
  - Informationen zum Zugriff auf die ONTAP Select Deploy API mit einem Browser . . . 8
  - Rufen Sie die Dokumentationsseite von ONTAP Select Deploy auf . . . 9
  - Einen ONTAP Select Deploy API-Aufruf verstehen und ausführen . . . 9
- Workflow-Prozesse . . . 10
  - Informationen zur Verwendung der ONTAP Select Deploy API Workflows . . . 10
  - Erstellung eines ONTAP Select Einzelknoten Evaluierungsclusters auf ESXi . . . 10
- Zugriff mit Python . . . 17
  - Informationen zum Zugriff auf die ONTAP Select Deploy API mit Python . . . 17
  - Informationen zu den Python-Skripten für ONTAP Select Deploy . . . 18
- Python-Codebeispiele . . . 19
  - Skript zum Erstellen eines ONTAP Select Clusters . . . 19
  - JSON für Skript zum Erstellen eines ONTAP Select Clusters . . . 26
  - Skript zum Hinzufügen einer ONTAP Select Knotenlizenz . . . 31
  - Skript zum Löschen eines ONTAP Select Clusters . . . 34
  - Gemeinsames Python-Unterstützungsmodul für ONTAP Select . . . 36
  - Skript zum Ändern der Größe von ONTAP Select Clusterknoten . . . 40

Automatisieren mit REST

Konzepte

Informationen zu REST-Webdiensten für die Bereitstellung und Verwaltung von ONTAP Select Clustern

Representational State Transfer (REST) ist ein Stil zur Entwicklung verteilter Webanwendungen. Angewendet auf das Design von Web-Service-APIs, etabliert REST eine Reihe von Technologien und Best Practices für die Bereitstellung serverbasierter Ressourcen und die Verwaltung ihrer Zustände. Es nutzt gängige Protokolle und Standards, um eine flexible Grundlage für die Bereitstellung und Verwaltung von ONTAP Select Clustern zu bieten.

Architektur und klassische Beschränkungen

REST wurde von Roy Fielding in seiner PhD "Dissertation" an der UC Irvine im Jahr 2000 formal artikuliert. Es definiert einen Architekturstil durch eine Reihe von Einschränkungen, die gemeinsam webbasierte Anwendungen und die zugrunde liegenden Protokolle verbessert haben. Die Einschränkungen etablieren eine RESTful-Webservices-Anwendung auf Basis einer Client/Server-Architektur unter Verwendung eines zustandslosen Kommunikationsprotokolls.

Ressourcen und Statusdarstellung

Ressourcen sind die grundlegenden Komponenten eines webbasierten Systems. Bei der Entwicklung einer REST-Webservices-Anwendung gehören folgende frühe Entwurfsaufgaben dazu:

- Identifizierung von System- oder serverbasierten Ressourcen. Jedes System nutzt und verwaltet Ressourcen. Eine Ressource kann eine Datei, eine Geschäftstransaktion, ein Prozess oder eine administrative Einheit 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 stets in einem von endlich vielen Zuständen. Die Zustände sowie die zugehörigen Operationen, die Zustandsänderungen bewirken, müssen klar definiert sein.

Nachrichten werden zwischen Client und Server ausgetauscht, um auf die Ressourcen zuzugreifen und ihren Zustand gemäß dem generischen CRUD-Modell (Create, Read, Update, and Delete) zu ändern.

URI-Endpunkte

Jede REST-Ressource muss definiert und mithilfe eines klar definierten Adressierungsschemas bereitgestellt werden. Die Endpunkte, an denen die Ressourcen lokalisiert und identifiziert werden, verwenden einen Uniform Resource Identifier (URI). Der URI bietet ein allgemeines Framework zur Erstellung eines eindeutigen Namens für jede Ressource im Netzwerk. Der Uniform Resource Locator (URL) ist eine Art URI, der in Webdiensten verwendet wird, um Ressourcen zu identifizieren und darauf zuzugreifen. Ressourcen werden typischerweise in einer hierarchischen Struktur ähnlich einem Dateiverzeichnis bereitgestellt.

HTTP-Nachrichten

HTTP (Hypertext Transfer Protocol) ist das Protokoll, das von Webdienst-Clients und -Servern zum Austausch von Anfrage- und Antwortnachrichten über die Ressourcen verwendet wird. Als Teil der Entwicklung einer

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

HTTP ist zustandslos. Um also eine Reihe zusammengehöriger Anfragen und Antworten einer Transaktion zuzuordnen, müssen zusätzliche Informationen in den HTTP-Headern enthalten sein, die mit den Anfrage-/Antwortdaten übertragen werden.

JSON-Formatierung

Informationen lassen sich auf verschiedene Weisen strukturieren und zwischen Client und Server übertragen. Die gängigste Option (und diejenige, die auch von der Deploy REST API verwendet wird) ist JavaScript Object Notation (JSON). JSON ist ein Industriestandard zur Darstellung einfacher Datenstrukturen in Klartext und dient der Übertragung von Zustandsinformationen, die Ressourcen beschreiben.

Informationen zum Zugriff auf die ONTAP Select Deploy API

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

Die in ONTAP Select Deploy enthaltene REST-API besitzt eine Versionsnummer. Diese Versionsnummer ist unabhängig von der Deploy-Versionsnummer. Das ONTAP Select 9.19.1 Deploy Verwaltungsprogramm beinhaltet Version 3 der REST-API.

Nativen Benutzeroberfläche für das Dienstprogramm bereitstellen

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

ONTAP Select Deploy Online-Dokumentationsseite

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

Benutzerdefiniertes Programm

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

Grundlegende Betriebsmerkmale der ONTAP Select Deploy API

REST etabliert zwar einen gemeinsamen Satz an Technologien und Best Practices, die Details der einzelnen APIs können jedoch je nach Design variieren. Sie sollten sich mit den Details und den betrieblichen Eigenschaften der ONTAP Select Deploy API vertraut machen, bevor Sie die API verwenden.

Hypervisor-Host versus ONTAP Select Node

Ein Hypervisor-Host ist die zentrale Hardwareplattform, auf der eine ONTAP Select virtuelle Maschine gehostet wird. Wenn eine virtuelle ONTAP Select-Maschine auf einem Hypervisor-Host bereitgestellt und aktiv ist, wird die virtuelle Maschine als *ONTAP Select-Knoten* betrachtet. Mit Version 3 der Deploy REST API sind die Host- und Knoten-Objekte 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.

Objektbezeichner

Jeder Ressourceninstanz bzw. jedem Objekt wird bei der Erstellung eine eindeutige Kennung zugewiesen. Diese Kennungen sind innerhalb einer bestimmten ONTAP Select Deploy Instanz global eindeutig. Nach einem API-Aufruf, der eine neue Objektinstanz erstellt, wird der zugehörige id-Wert im `location` Header der HTTP-Antwort an den Aufrufer zurückgegeben. Sie können die Kennung extrahieren und bei nachfolgenden Aufrufen verwenden, um auf die Ressourceninstanz zu verweisen.

> Der Inhalt und die interne Struktur der Objektbezeichner können sich jederzeit ändern. Sie sollten die Bezeichner nur bei den entsprechenden API-Aufrufen verwenden, wenn Sie auf die zugehörigen Objekte verweisen müssen.

Anforderungskennungen

Jeder erfolgreichen API-Anfrage wird eine eindeutige Kennung zugewiesen. Die Kennung wird im `request-id` Header der zugehörigen HTTP-Antwort zurückgegeben. Sie können eine Anfrage-Kennung verwenden, um gemeinsam auf die Aktivitäten einer einzelnen spezifischen API-Anfrage-Antwort-Transaktion zu verweisen. Beispielsweise können Sie alle Ereignismeldungen für eine Transaktion basierend auf der Anfrage-ID abrufen.

Synchrone und asynchrone Aufrufe

Es gibt zwei Hauptmethoden, mit denen ein Server eine vom Client empfangene HTTP (Hypertext Transfer Protocol)-Anfrage 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 bedeutet, dass der Server die Clientanfrage angenommen und einen Hintergrundprozess zur Bearbeitung gestartet hat. Ob die Anfrage erfolgreich war oder nicht, ist nicht sofort ersichtlich und muss durch weitere API-Aufrufe ermittelt werden.

Bestätigen Sie den Abschluss eines langwierigen Auftrags

Generell wird jede Operation, die lange dauern kann, asynchron mithilfe einer Hintergrundaufgabe auf dem Server verarbeitet. Mit der Deploy REST API ist jede Hintergrundaufgabe durch ein Job-Objekt verankert, das die Aufgabe verfolgt und Informationen wie den aktuellen Status bereitstellt. Ein Job-Objekt, einschließlich 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 weitere Möglichkeiten, den Erfolg oder Misserfolg einer Anfrage zu ermitteln, darunter:

- Ereignismeldungen Sie können alle Ereignismeldungen abrufen, die mit einem bestimmten API-Aufruf verknüpft sind, indem Sie die mit der ursprünglichen Antwort zurückgegebenen Anfrage-ID verwenden. Die Ereignismeldungen enthalten typischerweise einen Hinweis auf Erfolg oder Misserfolg und können auch beim Debuggen eines Fehlerzustands nützlich sein.
- Ressourcenzustand oder -status Mehrere der Ressourcen verwalten einen Zustand oder Statuswert, den Sie abfragen können, um indirekt den Erfolg oder Misserfolg einer Anfrage zu bestimmen.

Sicherheit

Die Deploy-API verwendet folgende Sicherheitstechnologien:

- Transport Layer Security: Der gesamte Netzwerkverkehr zwischen Deploy-Server und Client wird per TLS verschlüsselt. Die Verwendung des HTTP (Hypertext Transfer Protocol) über einen unverschlüsselten Kanal wird nicht unterstützt. TLS Version 1.2 wird unterstützt.
- Die HTTP-Authentifizierung verwendet die Basic-Authentifizierung für jede API-Transaktion. Jedem Request wird ein HTTP-Header hinzugefügt, der den Benutzernamen und das Passwort als Base64-String enthält.

Anfrage- und Antwort-API-Transaktion für ONTAP Select

Jeder Deploy-API-Aufruf wird als HTTP-Anfrage an die Deploy-virtuelle Maschine gesendet, die eine entsprechende Antwort an den Client generiert. Dieses Anfrage-/Antwort-Paar wird als API-Transaktion betrachtet. 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 steuern, wie ein API-Aufruf durch Parameter verarbeitet wird, die in der HTTP-Anfrage festgelegt sind.

Anfrageheader

Sie müssen mehrere Header in die HTTP-Anfrage einfügen, darunter:

- content-type Wenn der Anfragetext JSON enthält, muss dieser Header auf application/json gesetzt werden.
- accept Falls der Antworttext JSON enthalten soll, muss dieser Header auf application/json gesetzt werden.
- Die Basic-Authentifizierung muss mit dem Benutzernamen und Passwort erfolgen, die in einer Base64-Zeichenkette codiert sind.

Anfragekörper

Der Inhalt des Anfragetextes variiert je nach spezifischem Aufruf. Der HTTP-Anfragetext besteht aus einem der folgenden Elemente:

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

Filterobjekte

Bei einem API-Aufruf mit GET können Sie die zurückgegebenen Objekte anhand beliebiger Attribute

einschränken oder filtern. Beispielsweise können Sie einen exakten Wert angeben, der übereinstimmen soll:

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

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

Operator	Beschreibung
=	Gleich
<	Weniger als
>	Größer als
⇐	Kleiner oder gleich
>=	Größer oder gleich
	Oder
!	Nicht gleich
*	Gieriger Wildcard

Sie können auch eine Menge von Objekten zurückgeben, je nachdem, ob ein bestimmtes Feld gesetzt ist oder nicht, indem Sie das Schlüsselwort null oder seine Negation (!null) als Teil der Abfrage verwenden.

Objektfelder auswählen

Standardmäßig liefert ein API-Aufruf mit der GET-Methode nur die Attribute zurück, die das Objekt oder die Objekte eindeutig identifizieren. Dieser minimale Satz an Feldern dient als Schlüssel für jedes Objekt und variiert je nach Objekttyp. Sie können zusätzliche Objekteigenschaften mithilfe des Abfrageparameters fields auf folgende Weise auswählen:

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



Als Best Practice sollten Sie immer die spezifischen Felder identifizieren, die Sie benötigen. Sie sollten das Set an kostengünstigen oder teuren Feldern nur bei Bedarf abrufen. Die Einteilung in kostengünstige und teure Felder wird von NetApp basierend auf interner Leistungsanalyse bestimmt. Die Klassifizierung für ein bestimmtes Feld kann sich jederzeit ändern.

Sortieren Sie die Objekte 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 Feld „Typ“ in absteigender Reihenfolge und anschließend das Feld „ID“ in

aufsteigender Reihenfolge sortieren:
`order_by=type desc, id asc`

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

Paginierung

Beim Ausführen eines API-Aufrufs mit GET zum Zugriff auf eine Sammlung von Objekten desselben Typs werden standardmäßig alle übereinstimmenden Objekte zurückgegeben. Bei Bedarf können Sie die Anzahl der zurückgegebenen Datensätze mithilfe des Abfrageparameters max_records in der Anfrage begrenzen. Beispiel:
`max_records=20`

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

Sie können mehrere Anfragen senden, um die Ereignisse (oder beliebige Objekttypen) seitenweise zu durchlaufen. Jeder nachfolgende API-Aufruf sollte einen neuen Zeitwert basierend auf dem neuesten Ereignis im letzten Ergebnissatz verwenden.

Eine API-Antwort interpretieren

Jede API-Anfrage erzeugt eine Antwort, die an den Client zurückgesendet wird. Sie können die Antwort untersuchen, um festzustellen, ob sie erfolgreich war, und bei Bedarf zusätzliche Daten abrufen.

HTTP-Statuscode

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

Code	Bedeutung	Beschreibung
200	OK	Zeigt den Erfolg von Aufrufen an, die kein neues Objekt erzeugen.
201	Erstellt	Ein Objekt wurde erfolgreich erstellt; der Location-Response-Header enthält die eindeutige Kennung für das Objekt.
202	Akzeptiert	Zur Ausführung der Anfrage wurde ein Hintergrundprozess mit langer Laufzeit gestartet, der jedoch noch nicht abgeschlossen ist.
400	Ungültige Anforderung	Die Eingabe der Anfrage wird nicht erkannt oder ist ungeeignet.
403	Verboten	Der Zugriff wurde aufgrund eines Autorisierungsfehlers verweigert.
404	Nicht gefunden	Die in der Anfrage genannte Ressource existiert nicht.
405	Methode nicht zulässig	Das HTTP-Verb in der Anfrage wird für die Ressource nicht unterstützt.
409	Konflikt	Der Versuch, ein Objekt zu erstellen, ist fehlgeschlagen, da das Objekt bereits existiert.
500	Interner Fehler	Auf dem Server ist ein allgemeiner interner Fehler aufgetreten.
501	Nicht implementiert	Die URI ist bekannt, aber sie kann die Anfrage nicht ausführen.

Antwortheader

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

- request-id Jeder erfolgreichen API-Anfrage wird eine eindeutige request-id zugewiesen.
- location Wenn ein Objekt erstellt wird, enthält der Location-Header die vollständige URL zum neuen Objekt einschließlich der eindeutigen Objektkennung.

Antworttext

Der Inhalt der Antwort auf eine API-Anfrage variiert je nach Objekt, Verarbeitungstyp und Erfolg oder Misserfolg der Anfrage. Der Antworttext wird im JSON-Format dargestellt.

- Ein einzelnes Objekt kann mit einer Reihe von Feldern basierend auf der Anfrage zurückgegeben werden. Beispielsweise können Sie mit GET ausgewählte Eigenschaften eines Clusters anhand seiner eindeutigen Kennung abrufen.
- Es können mehrere Objekte aus einer Ressourcensammlung zurückgegeben werden. Dabei wird stets ein einheitliches Format verwendet, wobei `num_records` die Anzahl der Datensätze angegeben wird und die Datensätze ein Array der Objektinstanzen enthalten. Beispielsweise können Sie alle in einem bestimmten Cluster definierten Knoten abrufen.
- Wird ein API-Aufruf asynchron verarbeitet, wird ein Job-Objekt zurückgegeben, das die Hintergrundaufgabe steuert. Beispielsweise wird die POST-Anfrage zum Bereitstellen eines Clusters asynchron verarbeitet und gibt ein Job-Objekt zurück.
- Fehlerobjekt: Tritt ein Fehler auf, wird immer ein Fehlerobjekt zurückgegeben. Beispielsweise erhalten Sie einen Fehler, wenn Sie versuchen, einen Cluster mit einem Namen zu erstellen, der bereits existiert.
- In bestimmten Fällen werden keine Daten zurückgegeben und der Antworttext ist leer. Der Antworttext ist beispielsweise nach der Verwendung von DELETE zum Löschen eines vorhandenen Hosts leer.

Asynchrone Verarbeitung mit dem Job-Objekt für ONTAP Select

Einige Deploy-API-Aufrufe, insbesondere solche, die eine Ressource erstellen oder ändern, können länger dauern als andere Aufrufe. ONTAP Select Deploy verarbeitet diese lang andauernden Anfragen asynchron.

Asynchrone Anfragen werden mithilfe des Job-Objekts beschrieben.

Nach einem asynchronen API-Aufruf signalisiert der HTTP-Antwortcode 202, 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 weiterhin ausgeführt wird. Die Antwort enthält das Job-Objekt, das die Anfrage verankert, einschließlich seiner eindeutigen Kennung.

> Sie sollten die Online-Dokumentationsseite von ONTAP Select Deploy konsultieren, um festzustellen, welche API-Aufrufe asynchron ausgeführt werden.

Abfrage des Job-Objekts, das einer API-Anfrage zugeordnet ist

Das in der HTTP-Antwort zurückgegebene Job-Objekt enthält mehrere Eigenschaften. Sie können die Eigenschaft state abfragen, um festzustellen, ob die Anfrage erfolgreich abgeschlossen wurde. Ein Job-Objekt kann sich in einem der folgenden Zustände befinden:

- 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-Abfrage: Aktueller Jobstatus wird sofort zurückgegeben
- Der Auftragsstatus einer Long-Polling-Anfrage wird nur dann zurückgegeben, wenn eine der folgenden Bedingungen erfüllt ist:
  - Der Status hat sich nach dem im Abfragezeitpunkt angegebenen Datum/Uhrzeit geändert.
  - Zeitüberschreitungswert ist abgelaufen (1 bis 120 Sekunden)

Standard-Polling und Long-Polling verwenden denselben API-Aufruf, um ein Job-Objekt abzufragen. Eine Long-Polling-Anfrage enthält jedoch zwei Abfrageparameter: `poll_timeout` und `last_modified`.



Um die Arbeitslast auf der virtuellen Maschine Deploy zu reduzieren, sollten Sie stets Long Polling verwenden.

Allgemeines Verfahren zum Ausstellen einer asynchronen Anfrage

Sie können die folgende allgemeine Vorgehensweise verwenden, um einen asynchronen API-Aufruf durchzuführen:

1. Führe den asynchronen API-Aufruf aus.
2. Sie erhalten eine HTTP-Antwort 202, die die erfolgreiche Annahme der Anfrage signalisiert.
3. Extrahieren Sie die Kennung für das Job-Objekt aus dem Antworttext.
4. Führe innerhalb einer Schleife in jedem Durchlauf Folgendes aus:
   a. Ermitteln Sie den aktuellen Status des Jobs mit einer Long-Poll-Anfrage
   b. Befindet sich der Job in einem nicht-terminalen Zustand (in der Warteschlange, wird ausgeführt), führen Sie die Schleife erneut aus.
5. Beenden, wenn der Job einen Endzustand erreicht (Erfolg, Fehler).

Zugriff mit einem Browser

Informationen zum Zugriff auf die ONTAP Select Deploy API mit einem Browser

Bevor Sie die Online-Dokumentationsseite von Deploy verwenden, sollten Sie einige Dinge beachten.

Einsatzplan

Wenn Sie im Rahmen bestimmter Bereitstellungs- oder Verwaltungsaufgaben API-Aufrufe durchführen möchten, sollten Sie einen Bereitstellungsplan erstellen. Diese Pläne können formell oder informell sein und enthalten in der Regel Ihre Ziele sowie 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. Zusätzlich können Sie Details zum verwendeten JSON für API-Anfragen und -Antworten wie folgt anzeigen lassen:

- Beispielwert: Wenn Sie bei einem API-Aufruf *Beispielwert* auswählen, 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 *Modell* auswählen, wird eine vollständige Liste der JSON-Parameter mit einer Beschreibung für jeden Parameter angezeigt.

Vorsicht beim Ausführen von API-Aufrufen

Alle API-Operationen, die Sie über die Deploy-Dokumentationsseite durchführen, sind Live-Operationen. Sie sollten darauf achten, Konfigurationen oder andere Daten nicht versehentlich zu erstellen, zu aktualisieren oder zu löschen.

Rufen Sie die Dokumentationsseite von ONTAP Select Deploy auf

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

Bevor Sie beginnen

Sie benötigen Folgendes:

- IP-Adresse oder Domänenname der ONTAP Select Deploy-virtuellen Maschine
- Benutzername und Passwort für den Administrator

Schritte

1. Geben Sie die URL in Ihren Browser ein und wählen Sie **Eingabetaste**:

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

2. Die Anmeldung erfolgt mit dem Administrator-Benutzernamen und dem Passwort.

Ergebnis

Die Webseite mit der Deploy-Dokumentation wird angezeigt, wobei die Aufrufe am unteren Rand der Seite nach Kategorien geordnet sind.

Einen ONTAP Select Deploy API-Aufruf verstehen und ausführen

Die Details aller API-Aufrufe sind auf der ONTAP Select Deploy Online-Dokumentationswebseite in einem einheitlichen Format dokumentiert und dargestellt. Durch das Verständnis eines einzelnen API-Aufrufs können Sie die Details aller API-Aufrufe abrufen und interpretieren.

Bevor Sie beginnen

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

Über diese Aufgabe

Sie können die Konfigurationsinformationen, die einen ONTAP Select Cluster beschreiben, anhand seiner eindeutigen Kennung abrufen. In diesem Beispiel werden alle als kostengünstig klassifizierten Felder zurückgegeben. Es empfiehlt sich jedoch, nur die spezifischen Felder anzufordern, die benötigt werden.

Schritte

1. Scrollen Sie auf der Hauptseite bis ganz nach unten und wählen Sie **Cluster** aus.
2. Wählen Sie **GET /clusters/{cluster_id}**, um die Details des API-Aufrufs anzuzeigen, der verwendet wurde, um Informationen über einen ONTAP Select Cluster abzurufen.

Workflow-Prozesse

Informationen zur Verwendung der ONTAP Select Deploy API Workflows

Sie sollten sich darauf vorbereiten, die Arbeitsabläufe zu überprüfen und anzuwenden.

Die in den Workflows verwendeten API-Aufrufe verstehen

Die ONTAP Select Online-Dokumentationsseite enthält die Details zu jedem REST-API-Aufruf. Anstatt diese Details hier zu wiederholen, sind bei jedem in den Workflow-Beispielen verwendeten API-Aufruf nur die Informationen enthalten, die Sie benötigen, um den Aufruf auf der Dokumentationsseite zu finden. Nachdem Sie einen bestimmten API-Aufruf gefunden haben, können Sie die vollständigen Details des Aufrufs einsehen, einschließlich der Eingabeparameter, Ausgabeformate, HTTP-Statuscodes und des Anfrageverarbeitungstyps.

Die folgenden Informationen sind für jeden API-Aufruf innerhalb eines Workflows enthalten, um den Aufruf auf der Dokumentationsseite zu finden:

- Die API-Aufrufe sind auf der Dokumentationsseite in funktional verwandte Bereiche oder Kategorien unterteilt. Um einen bestimmten API-Aufruf zu finden, scrollen Sie zum Seitenende und wählen Sie die entsprechende API-Kategorie aus.
- Das HTTP-Verb identifiziert die Aktion, die an einer Ressource ausgeführt wird. Jeder API-Aufruf wird über ein einzelnes HTTP-Verb ausgeführt.
- Der Pfad bestimmt die spezifische Ressource, auf die sich die Aktion im Rahmen eines Aufrufs bezieht. Die Pfadzeichenfolge wird an die Kern-URL angehängt, um die vollständige URL zu bilden, die die Ressource identifiziert.

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

Zusätzlich zur ONTAP Select-Dokumentationsseite können Sie die Deploy REST API auch direkt über eine Programmiersprache wie Python aufrufen. In diesem Fall unterscheidet sich die Kern-URL geringfügig von der URL, die Sie für den Zugriff auf die Online-Dokumentationsseite verwenden. Beim direkten Zugriff auf die API müssen Sie /api an die Domain- und Port-Zeichenfolge anhängen. Beispiel:
`http://deploy.mycompany.com/api`

Erstellung eines ONTAP Select Einzelknoten Evaluierungsclusters auf ESXi

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

Der Workflow zur Clustererstellung unterscheidet sich in folgenden Situationen:

- Der ESXi-Host wird nicht von vCenter verwaltet (eigenständiger Host).
- Innerhalb des Clusters werden mehrere Nodes oder Hosts verwendet.
- Der Cluster ist in einer Produktionsumgebung mit einer erworbenen Lizenz implementiert.
- Anstelle von VMware ESXi wird der KVM Hypervisor verwendet.

Schritt 1: vCenter Server-Anmeldeinformationen registrieren

Bei der Bereitstellung auf einem von einem vCenter Server verwalteten ESXi-Host ist es erforderlich, vor der Registrierung des Hosts Anmeldeinformationen hinzuzufügen. Das Dienstprogramm Deploy kann dann die Anmeldeinformationen zur Authentifizierung bei vCenter verwenden.

Kategorie	HTTP-Verb	Pfad
Einsetzen	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-Eingabe (Schritt 01)

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

Verarbeitungstyp

asynchron

Ausgabe

- Anmeldeinformations-ID im Location-Antwortheader
- Jobobjekt

Schritt 2: Registrierung eines Hypervisor-Hosts

Einen Hypervisor-Host hinzufügen, auf dem die virtuelle Maschine mit dem ONTAP Select Node ausgeführt wird.

Kategorie	HTTP-Verb	Pfad
Cluster	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-Eingabe (step02)

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

Verarbeitungstyp

asynchron

Ausgabe

- Host-ID im Location-Antwortheader
- Jobobjekt

Schritt 3: Einen Cluster erstellen

Wenn ein ONTAP Select Cluster erstellt wird, wird die grundlegende Clusterkonfiguration registriert und Deploy generiert die Node-Namen automatisch.

Kategorie	HTTP-Verb	Pfad
Cluster	POST	/clusters

Curl

Der Abfrageparameter node_count ist für einen Einzelknotencluster auf 1 zu setzen.

```
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 Standortantwort-Header

Schritt 4: Das Cluster konfigurieren

Im Rahmen der Clusterkonfiguration müssen Sie die folgenden Attribute angeben.

Kategorie	HTTP-Verb	Pfad
Cluster	PATCH	/clusters/{cluster_id}

Curl

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

Schritt 5: Node-Name abrufen

Das Deploy Verwaltungsprogramm generiert beim Erstellen eines Clusters automatisch die Node-IDs und Node-Namen. Vor der Konfiguration eines Node muss die zugewiesene ID abgerufen werden.

Kategorie	HTTP-Verb	Pfad
Cluster	GET	/clusters/{cluster_id}/nodes

Curl

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 Node mit der eindeutigen ID und dem Node-Name beschreiben

Schritt 6: Die Nodes konfigurieren

Die Basiskonfiguration für den Node bereitstellen, dies ist der erste von drei API-Aufrufen, die zur Konfiguration eines Node verwendet werden.

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

Curl

Die Cluster-ID und die Node-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)

Die Host-ID angeben, auf der der ONTAP Select Node ausgeführt wird.

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

Verarbeitungstyp

Synchron

Ausgabe

Keine

Schritt 7: Knotennetzwerke abrufen

Die Daten- und Managementnetzwerke, die vom Node im Single-Node-Cluster verwendet werden, sind zu identifizieren. Das interne Netzwerk wird bei einem Single-Node-Cluster nicht verwendet.

Kategorie	HTTP-Verb	Pfad
Cluster	GET	/clusters/{cluster_id}/nodes/{node_id}/networks

Curl

Die Cluster-ID und die Node-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 Node-Name beschreiben, einschließlich der eindeutigen ID und des Zwecks

Schritt 8: Node-Netzwerk konfigurieren

Die Daten- und Managementnetzwerke werden konfiguriert. Das interne Netzwerk wird bei einem Single-Node-Cluster nicht verwendet.

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

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

Curl

Geben Sie die Cluster-ID, die Node-ID und die Netzwerk-ID an.

```
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)

Den Namen des Netzwerks angeben.

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

Verarbeitungstyp

Synchron

Ausgabe

Keine

Schritt 9: Konfiguration des Node Storage Pools

Der letzte Schritt bei der Konfiguration eines Knotens ist das Anbinden eines Speicherpools. Die verfügbaren Speicherpools können Sie über den vSphere Webclient oder optional über die Deploy REST API ermitteln.

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

Curl

Geben Sie die Cluster-ID, die Node-ID und die Netzwerk-ID an.

```
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 Speicherkapazität des Pools beträgt 2 TB.

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

Verarbeitungstyp

Synchron

Ausgabe

Keine

Schritt 10: Cluster bereitstellen

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

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

Curl

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)

Das Passwort für das ONTAP Administrator-Konto angeben.

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

Verarbeitungstyp

asynchron

Ausgabe

- Jobobjekt

Verwandte Informationen

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

Zugriff mit Python

Informationen zum Zugriff auf die ONTAP Select Deploy API mit Python

Sie müssen die Umgebung vorbereiten, bevor Sie die Beispiel-Python-Skripte ausführen.

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

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

Darüber hinaus benötigen Sie folgende Informationen:

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

Informationen zu den Python-Skripten für ONTAP Select Deploy

Die Beispiel-Python-Skripte ermöglichen die Ausführung verschiedener Aufgaben. Sie sollten die Skripte verstehen, bevor Sie sie in einer Live-Deploy-Instanz verwenden.

Gemeinsame Konstruktionsmerkmale

Die Skripte wurden mit folgenden gemeinsamen Merkmalen entworfen:

- Führen Sie die Ausführung über die Befehlszeilenschnittstelle eines Client-Rechners durch. Sie können die Python-Skripte von jedem entsprechend konfigurierten Client-Rechner aus ausführen. Weitere Informationen finden Sie unter *Bevor Sie beginnen*.
- Akzeptieren von CLI-Eingabeparametern. Jedes Skript wird über Eingabeparameter in der CLI gesteuert.
- Eingabedatei lesen: Jedes Skript liest eine Eingabedatei entsprechend seinem Zweck. Beim Erstellen oder Löschen eines Clusters muss eine JSON-Konfigurationsdatei angegeben werden. Beim Hinzufügen einer Knotenlizenz muss eine gültige Lizenzdatei angegeben werden.
- Verwenden Sie ein gemeinsames Support-Modul. Das gemeinsame Support-Modul *deploy_requests.py* enthält eine einzelne Klasse. Es wird von jedem der Skripte importiert und verwendet.

Erstellen Sie einen Cluster

Sie können einen ONTAP Select Cluster mithilfe des Skripts 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: Die Bereitstellung kann auf ESXi oder KVM erfolgen (abhängig von der Deploy-Version). Bei der Bereitstellung auf ESXi kann der Hypervisor von vCenter verwaltet oder als eigenständiger Host betrieben werden.
- Clustergröße Sie können einen Single-Node- oder Multi-Node-Cluster bereitstellen.
- Evaluierungs- oder Produktionslizenz Sie können einen Cluster mit einer Evaluierungs- oder einer erworbenen Lizenz für die Produktion bereitstellen.

Die CLI-Eingabeparameter für das Skript umfassen:

- Hostname oder IP-Adresse des Deploy-Servers
- Passwort für das Administratorkonto
- Name der JSON-Konfigurationsdatei
- Ausführliche Meldungsausgabe

Fügen Sie eine Knotenlizenz hinzu

Wenn Sie einen Produktionscluster bereitstellen, müssen Sie für jeden Knoten eine Lizenz mithilfe des Skripts *add_license.py* 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 Administratorkonto
- Name der Lizenzdatei

- ONTAP-Benutzername mit Berechtigung zum Hinzufügen der Lizenz
- Passwort für den ONTAP Benutzer

Einen Cluster löschen

Sie können einen bestehenden 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 Administratorkonto
- Name der JSON-Konfigurationsdatei

Python-Codebeispiele

Skript zum Erstellen eines ONTAP Select Clusters

Sie können das folgende Skript verwenden, um einen Cluster auf Basis von 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 dem Skript eine JSON-Datei als Eingabe bereitstellen. Sie können das passende JSON-Beispiel entsprechend Ihren Bereitstellungsplänen kopieren und anpassen.

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 Node 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 bestehenden 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 Python-Unterstützungsmodul 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())
```

Copyright-Informationen

Copyright © 2026 NetApp. Alle Rechte vorbehalten. Gedruckt in den USA. Dieses urheberrechtlich geschützte Dokument darf ohne die vorherige schriftliche Genehmigung des Urheberrechtsinhabers in keiner Form und durch keine Mittel – weder grafische noch elektronische oder mechanische, einschließlich Fotokopieren, Aufnehmen oder Speichern in einem elektronischen Abrufsystem – auch nicht in Teilen, vervielfältigt werden.

Software, die von urheberrechtlich geschütztem NetApp Material abgeleitet wird, unterliegt der folgenden Lizenz und dem folgenden Haftungsausschluss:

DIE VORLIEGENDE SOFTWARE WIRD IN DER VORLIEGENDEN FORM VON NETAPP ZUR VERFÜGUNG GESTELLT, D. H. OHNE JEGLICHE EXPLIZITE ODER IMPLIZITE GEWÄHRLEISTUNG, EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF DIE STILLSCHWEIGENDE GEWÄHRLEISTUNG DER MARKTGÄNGIGKEIT UND EIGNUNG FÜR EINEN BESTIMMTEN ZWECK, DIE HIERMIT AUSGESCHLOSSEN WERDEN. NETAPP ÜBERNIMMT KEINERLEI HAFTUNG FÜR DIREKTE, INDIREKTE, ZUFÄLLIGE, BESONDERE, BEISPIELHAFTE SCHÄDEN ODER FOLGESCHÄDEN (EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF DIE BESCHAFFUNG VON ERSATZWAREN ODER -DIENSTLEISTUNGEN, NUTZUNGS-, DATEN- ODER GEWINNVERLUSTE ODER UNTERBRECHUNG DES GESCHÄFTSBETRIEBS), UNABHÄNGIG DAVON, WIE SIE VERURSACHT WURDEN UND AUF WELCHER HAFTUNGSTHEORIE SIE BERUHEN, OB AUS VERTRAGLICH FESTGELEGTER HAFTUNG, VERSCHULDENSUNABHÄNGIGER HAFTUNG ODER DELIKTSHAFTUNG (EINSCHLIESSLICH FAHRLÄSSIGKEIT ODER AUF ANDEREM WEGE), DIE IN IRGENDEINER WEISE AUS DER NUTZUNG DIESER SOFTWARE RESULTIEREN, SELBST WENN AUF DIE MÖGLICHKEIT DERARTIGER SCHÄDEN HINGEWIESEN WURDE.

NetApp behält sich das Recht vor, die hierin beschriebenen Produkte jederzeit und ohne Vorankündigung zu ändern. NetApp übernimmt keine Verantwortung oder Haftung, die sich aus der Verwendung der hier beschriebenen Produkte ergibt, es sei denn, NetApp hat dem ausdrücklich in schriftlicher Form zugestimmt. Die Verwendung oder der Erwerb dieses Produkts stellt keine Lizenzierung im Rahmen eines Patentrechts, Markenrechts oder eines anderen Rechts an geistigem Eigentum von NetApp dar.

Das in diesem Dokument beschriebene Produkt kann durch ein oder mehrere US-amerikanische Patente, ausländische Patente oder anhängige Patentanmeldungen geschützt sein.

ERLÄUTERUNG ZU „RESTRICTED RIGHTS“: Nutzung, Vervielfältigung oder Offenlegung durch die US-Regierung unterliegt den Einschränkungen gemäß Unterabschnitt (b)(3) der Klausel „Rights in Technical Data – Noncommercial Items“ in DFARS 252.227-7013 (Februar 2014) und FAR 52.227-19 (Dezember 2007).

Die hierin enthaltenen Daten beziehen sich auf ein kommerzielles Produkt und/oder einen kommerziellen Service (wie in FAR 2.101 definiert) und sind Eigentum von NetApp, Inc. Alle technischen Daten und die Computersoftware von NetApp, die unter diesem Vertrag bereitgestellt werden, sind gewerblicher Natur und wurden ausschließlich unter Verwendung privater Mittel entwickelt. Die US-Regierung besitzt eine nicht ausschließliche, nicht übertragbare, nicht unterlizenzierbare, weltweite, limitierte unwiderrufliche Lizenz zur Nutzung der Daten nur in Verbindung mit und zur Unterstützung des Vertrags der US-Regierung, unter dem die Daten bereitgestellt wurden. Sofern in den vorliegenden Bedingungen nicht anders angegeben, dürfen die Daten ohne vorherige schriftliche Genehmigung von NetApp, Inc. nicht verwendet, offengelegt, vervielfältigt, geändert, aufgeführt oder angezeigt werden. Die Lizenzrechte der US-Regierung für das US-Verteidigungsministerium sind auf die in DFARS-Klausel 252.227-7015(b) (Februar 2014) genannten Rechte beschränkt.

Markeninformationen

NETAPP, das NETAPP Logo und die unter http://www.netapp.com/TM aufgeführten Marken sind Marken von NetApp, Inc. Andere Firmen und Produktnamen können Marken der jeweiligen Eigentümer sein.