Automatisierung mit REST

ONTAP Select

NetApp
November 21, 2024

Inhalt

Automatisierung mit REST

Konzepte

REST-Web-Services-Grundlage

Representational State Transfer (REST) ist ein Stil für die Erstellung von verteilten Web-Anwendungen. Bei der Anwendung auf das Design einer Web-Services-API werden eine Reihe von Technologien und Best Practices erstellt, um serverbasierte Ressourcen freizulegen und deren Status zu verwalten. Die Technologie nutzt Mainstream-Protokolle und -Standards, um eine flexible Grundlage für die Bereitstellung und das Management von ONTAP Select Clustern zu bieten.

Architektur und klassische Einschränkungen

REST wurde formell von Roy Fielding in seinem PhD artikuliert "Dissertation" An der UC Irvine im Jahr 2000. Es definiert einen Architekturstil durch eine Reihe von Einschränkungen, die gemeinsam Web-basierte Anwendungen und die zugrunde liegenden Protokolle verbessert haben. Durch diese Einschränkungen wird eine RESTful Web Services-Applikation basierend auf einer Client-/Serverarchitektur mit einem statusfreien Kommunikationsprotokoll hergestellt.

Ressourcen- und Zustandsdarstellung

Ressourcen sind die Grundkomponenten eines webbasierten Systems. Beim Erstellen einer ANWENDUNG FÜR REST-Webservices umfassen die frühen Designaufgaben Folgendes:

- Identifizierung von System- oder serverbasierten Ressourcen
  Jedes System nutzt und verwaltet Ressourcen. Eine Ressource kann eine Datei-, Geschäftstransaktion-, Prozess- oder Verwaltungseinheit sein. Eine der ersten Aufgaben bei der Entwicklung einer auf REST-Webservices basierenden Applikation ist die Identifizierung der Ressourcen.
- Definition von Ressourcenstatus und zugehörigen Statusoperationen
  Die Ressourcen befinden sich immer in einer endlichen Anzahl von Staaten. Die Zustände sowie die damit verbundenen Operationen, die zur Auswirkung der Statusänderungen verwendet werden, müssen klar definiert werden.

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

URI-Endpunkte

Jede REST-Ressource muss definiert und über ein gut definiertes Adressierungssystem verfügbar gemacht werden. Die Endpunkte, in denen die Ressourcen gefunden und identifiziert werden, verwenden einen einheitlichen Resource Identifier (URI). Der URI bietet ein allgemeines Framework zum Erstellen eines eindeutigen Namens für jede Ressource im Netzwerk. Der Uniform Resource Locator (URL) ist ein URI-Typ, der mit Webservices zur Identifizierung und zum Zugriff von Ressourcen verwendet wird. Ressourcen werden in der Regel in einer hierarchischen Struktur ausgesetzt, die einem Dateiverzeichnis ähnelt.

HTTP-Meldungen

Hypertext Transfer Protocol (HTTP) ist das Protokoll, das vom Webservice-Client und -Server zum Austausch von Anforderungs- und Antwortmeldungen zu den Ressourcen verwendet wird. Im Rahmen der Entwicklung

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

HTTP ist statusfrei. Um eine Reihe verwandter Anforderungen und Antworten innerhalb einer Transaktion zuzuordnen, müssen zusätzliche Informationen in die HTTP-Header enthalten sein, die mit den Request/Response-Datenströmen übertragen werden.

JSON-Formatierung

Während Informationen auf verschiedene Weise zwischen Client und Server strukturiert und übertragen werden können, ist die beliebteste Option (und die bei der REST-API implementieren verwendete Option) JavaScript Object Notation (JSON). JSON ist ein Branchenstandard für die Darstellung einfacher Datenstrukturen im Klartext und wird zur Übertragung von Zustandsdaten zur Beschreibung der Ressourcen verwendet.

So erhalten Sie Zugriff auf die Bereitstellungs-API

Aufgrund der inhärenten Flexibilität VON REST-Webservices ist der Zugriff auf die ONTAP Select Deploy API auf verschiedene Weise möglich.

Implementieren Sie die native Benutzeroberfläche von Utility

Der primäre Weg, um auf die API zuzugreifen, ist über die ONTAP Select Deploy Web-Benutzeroberfläche. Der Browser ruft die API auf und formatiert die Daten entsprechend dem Design der Benutzeroberfläche neu. Sie haben auch Zugriff auf die API über die Befehlszeilenschnittstelle von Deploy Utility.

ONTAP Select Seite zur Online-Dokumentation bereitstellen

Die Seite ONTAP Select Online-Dokumentation bereitstellen stellt bei Verwendung eines Browsers einen alternativen Zugriffspunkt dar. Die Seite bietet nicht nur die Möglichkeit, einzelne API-Aufrufe direkt auszuführen, sondern enthält auch eine detaillierte Beschreibung der API, einschließlich Eingabeparameter und anderer Optionen für jeden Aufruf. Die API-Aufrufe sind in verschiedene funktionale Bereiche oder Kategorien gegliedert.

Benutzerdefiniertes Programm

Die Bereitstellungs-API kann über eine von mehreren verschiedenen Programmiersprachen und Tools auf die Bereitstellungsschnittstelle zugegriffen werden. Beliebte Optionen sind Python, Java und Curl. Ein Programm, Skript oder Tool, das die API verwendet, fungiert als REST-Web-Services-Client. Mithilfe einer Programmiersprache lernen Sie die API besser kennen und erhalten die Möglichkeit, die ONTAP Select Implementierungen zu automatisieren.

Implementieren der API-Versionierung

Der REST API, die in ONTAP Select Deploy enthalten ist, wird eine Versionsnummer zugewiesen. Die API-Versionsnummer ist unabhängig von der Versionsnummer für die Bereitstellung. Sie sollten die API-Version kennen, die in Ihrer Version von Deploy enthalten ist, und wissen, welche Auswirkungen dies auf Ihre Verwendung der API hat.

Die aktuelle Version des Deploy Administration Utility enthält Version 3 DER REST API. Frühere Versionen des Deploy Utility umfassen die folgenden API-Versionen:

Implementierung 2.8 und höher

ONTAP Select Deploy 2.8. Alle neueren Versionen enthalten Version 3 der REST API.

Implementierung 2.7.2 und früher

ONTAP Select Deploy 2.7.2; alle älteren Versionen enthalten Version 2 DER REST API.

Die Versionen 2 und 3 DER REST-API sind nicht kompatibel. Wenn Sie nach einem früheren Release, das Version 2 der API enthält, ein Upgrade auf die Bereitstellung von 2.8 oder höher durchführen, müssen Sie jeden vorhandenen Code aktualisieren, der direkt auf die API zugreift, sowie alle Skripte über die Befehlszeilenschnittstelle.

Grundlegende betriebliche Eigenschaften

IM RUHEZUSTAND werden einheitliche Technologien und Best Practices erstellt, jedoch können die Details jeder API je nach dem verfügbaren Design variieren. Vor der Verwendung der API sollten Sie auf die Details und betrieblichen Merkmale der ONTAP Select Deploy-API achten.

Hypervisor-Host oder ONTAP Select-Node

Ein *Hypervisor-Host* ist die zentrale Hardware-Plattform, die eine ONTAP Select Virtual Machine hostet. Wenn eine ONTAP Select Virtual Machine auf einem Hypervisor-Host bereitgestellt und aktiv ist, gilt die Virtual Machine als „*ONTAP Select Node*“. Ab Version 3 der Deploy REST API sind Host- und Node-Objekte voneinander getrennt und unterscheiden sich. Dadurch wird eine 1:n-Beziehung möglich, bei der ein oder mehrere ONTAP Select Nodes auf demselben Hypervisor-Host ausgeführt werden können.

Objektkennungen

Jeder Ressourceninstanz oder jedem Objekt wird eine eindeutige Kennung zugewiesen, wenn sie erstellt wird. Diese Kennungen sind global eindeutig in einer bestimmten Instanz von ONTAP Select Deploy. Nachdem ein API-Aufruf ausgegeben wurde, der eine neue Objektinstanz erstellt, wird der zugeordnete id-Wert an den Anrufer im zurückgegeben `location` Kopfzeile der HTTP-Antwort. Sie können die Kennung extrahieren und bei nachfolgenden Anrufen verwenden, wenn Sie sich auf die Ressourceninstanz beziehen.

Der Inhalt und die interne Struktur der Objektkennungen können jederzeit geändert werden. Wenn Sie auf die zugeordneten Objekte verweisen, sollten Sie die Kennungen für die entsprechenden API-Aufrufe nur nach Bedarf verwenden.

Identifikatoren anfordern

Jeder erfolgreichen API-Anforderung wird eine eindeutige Kennung zugewiesen. Die Kennung wird im zurückgegeben `request-id` Kopfzeile der zugehörigen HTTP-Antwort. Sie können eine Anforderungskennung verwenden, um sich kollektiv auf die Aktivitäten einer einzelnen spezifischen API-Anforderungstransaktion zu beziehen. Sie können beispielsweise alle Ereignismeldungen einer Transaktion basierend auf der Anfrage-ID abrufen

Synchrone und asynchrone Anrufe

Es gibt zwei primäre Möglichkeiten, wie ein Server eine von einem Client empfangene HTTP-Anfrage durchführt:

- Synchron
  Der Server führt die Anforderung 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 Clientanforderung angenommen hat und eine Hintergrundaufgabe gestartet hat, um die Anforderung abzuschließen. Der endgültige Erfolg oder Fehler ist nicht sofort verfügbar und muss durch zusätzliche API-Aufrufe ermittelt werden.

Bestätigen Sie den Abschluss eines lang laufenden Jobs

Im Allgemeinen wird jede Operation, die lange Zeit in Anspruch nehmen kann, asynchron mit einem verarbeitet Hintergrundaufgabe auf dem Server. Mit der Deploy REST API wird jede Hintergrundaufgabe durch ein verankert
Jobobjekt, das die Aufgabe verfolgt und Informationen bereitstellt, z. B. den aktuellen Status. Ein Job-Objekt, Einschließlich seiner eindeutigen Kennung wird in der HTTP-Antwort zurückgegeben, nachdem eine Hintergrundaufgabe erstellt wurde.

Sie können das Jobobjekt direkt abfragen, um den Erfolg oder den Fehler des zugeordneten API-Aufrufs zu ermitteln.
Weitere Informationen finden Sie unter „*Asynchronous Processing Using the Job Object*“.

Neben der Verwendung des Objekts Job gibt es weitere Möglichkeiten, wie Sie den Erfolg oder das Scheitern eines bestimmen können
Anforderung, einschließlich:

- Ereignismeldungen
  Sie können alle Ereignismeldungen, die einem bestimmten API-Aufruf zugeordnet sind, mithilfe der mit der ursprünglichen Antwort zurückgegebenen Anforderungs-id abrufen. Die Ereignismeldungen enthalten in der Regel Hinweise auf Erfolg oder Fehler und können auch nützlich sein, wenn ein Fehlerzustand behoben wird.
- Ressourcenstatus oder -Status
  Mehrere der Ressourcen behalten einen Status oder Statuswert bei, den Sie abfragen können, um indirekt den Erfolg oder das Fehlschlagen einer Anfrage zu bestimmen.

Sicherheit

Die Deploy-API nutzt die folgenden Sicherheitstechnologien:

- Sicherheit In Transportschicht
  Der gesamte Datenverkehr, der zwischen dem bereitzustellenden Server und dem Client über das Netzwerk gesendet wird, 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 Anforderung wird ein HTTP-Header hinzugefügt, der den Benutzernamen und das Passwort in einem base64-String enthält.

API-Transaktion bei Anfrage und Reaktion

Jeder API-Aufruf zur Bereitstellung wird als HTTP-Anforderung an die virtuelle Maschine Bereitstellen ausgeführt, die eine entsprechende Antwort auf den Client erzeugt. Dieses Anforderungs-/Antwortpaar wird als API-Transaktion betrachtet. Bevor Sie die Deploy-API

verwenden, sollten Sie mit den zur Steuerung einer Anfrage verfügbaren Eingabevariablen und dem Inhalt der Antwortausgabe vertraut sein.

Eingabevariablen, die eine API-Anforderung steuern

Sie können steuern, wie ein API-Aufruf über in der HTTP-Anforderung festgelegte Parameter verarbeitet wird.

Anfragekopfzeilen

In der HTTP-Anfrage müssen mehrere Header enthalten sein, darunter:

- Content-Typ
  Wenn der Anforderungskörper 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 Passwort in einem base64-String eingerichtet werden.

Text anfordern

Der Inhalt der Anfraentext variiert je nach Anruf. Der HTTP-Request-Text besteht aus einem der folgenden Elemente:

- JSON-Objekt mit Eingabevariablen (z. B. der Name eines neuen Clusters)
- Leer

Objekte filtern

Wenn Sie einen API-Aufruf ausgeben, der GET verwendet, können Sie die zurückgegebenen Objekte anhand eines beliebigen Attributs einschränken oder filtern. Sie können beispielsweise einen genauen Wert angeben, der übereinstimmt:

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

Zusätzlich zu einer genauen Übereinstimmung stehen anderen Operatoren zur Verfügung, um einen Satz von Objekten über einen Wertebereich zurückzugeben. ONTAP Select unterstützt die unten aufgeführten Filteroperatoren.

Operator	Beschreibung
=	Gleich
<	Kleiner als
>	Größer als
≪=	Kleiner oder gleich
>=	Größer oder gleich
	Oder
!	Nicht gleich
*	Gierige Wildcard

Sie können auch einen Satz von Objekten zurückgeben, basierend darauf, ob ein bestimmtes Feld gesetzt wird oder nicht, indem Sie das Null-Schlüsselwort oder dessen Negation (!null) als Teil der Abfrage verwenden.

Auswählen von Objektfeldern

Standardmäßig gibt die Ausgabe eines API-Aufrufs mithilfe VON GET nur die Attribute zurück, die das Objekt oder die Objekte eindeutig identifizieren. Dieser minimale Feldsatz dient als Schlüssel für jedes Objekt und variiert je nach Objekttyp. Sie können zusätzliche Objekteigenschaften mithilfe des Abfrageparameters Felder wie folgt auswählen:

- Günstige Felder
  Angeben `fields=*` Zum Abrufen der Objektfelder, die im lokalen Serverspeicher verwaltet werden oder für den Zugriff nur wenig verarbeitet werden müssen.
- Teure Felder
  Angeben `fields=**` Zum Abrufen aller Objektfelder, einschließlich solcher, die für den Zugriff zusätzliche Serververarbeitung erforderlich sind.
- Benutzerdefinierte Feldauswahl
  Nutzung `fields=FIELDNAME` Um das genaue Feld anzugeben, das Sie wünschen. Wenn Sie mehrere Felder anfordern, müssen die Werte durch Kommas ohne Leerzeichen getrennt werden.

Als Best Practice sollten Sie immer die gewünschten Felder identifizieren. Sie sollten nur die Reihe von kostengünstigen oder teuren Feldern abrufen, wenn Sie benötigen. Die kostengünstige und teure Klassifizierung wird durch NetApp auf der Grundlage interner Leistungsanalysen festgelegt. Die Klassifizierung für ein bestimmtes Feld kann sich jederzeit ändern.

Objekte im Ausgabesatz sortieren

Die Datensätze in einer Ressourcensammlung werden in der vom Objekt definierten Standardreihenfolge zurückgegeben. Sie können die Reihenfolge mit dem Abfrageparameter 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, gefolgt von id in aufsteigender Reihenfolge sortieren:

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

Wenn Sie mehrere Parameter eingeben, müssen Sie die Felder mit einem Komma trennen.

Paginierung

Wenn Sie einen API-Aufruf über GET ausgeben, um auf eine Sammlung von Objekten desselben Typs zuzugreifen, werden alle übereinstimmenden Objekte standardmäßig zurückgegeben. Bei Bedarf können Sie die Anzahl der zurückgegebenen Datensätze mithilfe des Abfrageparameters max_Records mit der Anforderung begrenzen. Beispiel:

```
max_records=20
```

Bei Bedarf können Sie diesen Parameter mit anderen Abfrageparametern kombinieren, um den Ergebnissatz einzugrenzen. Beispiel: Im Folgenden werden bis zu 10 Systemereignisse angezeigt, die nach der angegebenen Zeit generiert wurden:

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

Sie können mehrere Anfragen zur Seite über die Ereignisse (oder jeden Objekttyp) ausgeben. Jeder

nachfolgende API-Aufruf sollte einen neuen Zeitwert verwenden, der auf dem letzten Ereignis des letzten Ergebnisset basiert.

Eine API-Antwort interpretieren

Jede API-Anfrage generiert eine Antwort an den Client. Sie können die Antwort prüfen, um festzustellen Ob die Daten erfolgreich waren und zusätzliche Daten nach Bedarf abgerufen werden konnten.

HTTP-Statuscode

Im Folgenden werden die von der REST-API „Bereitstellen“ verwendeten HTTP-Statuscodes beschrieben.

Codieren	Bedeutung	Beschreibung
200	OK	Zeigt Erfolg für Anrufe an, die kein neues Objekt erstellen.
201	Erstellt	Ein Objekt wurde erfolgreich erstellt. Der Header für die Standortantwort enthält die eindeutige Kennung für das Objekt.
202	Akzeptiert	Ein schon seit langem laufender Hintergrundjob wurde gestartet, um die Anforderung auszuführen, der Vorgang wurde jedoch noch nicht abgeschlossen.
400	Schlechte Anfrage	Die Eingabe der Anfrage ist nicht erkannt oder nicht angemessen.
403	Verboten	Der Zugriff wird aufgrund eines Autorisierungsfehlers verweigert.
404	Nicht gefunden	Die Ressource, auf die in diesem Antrag verwiesen wird, ist nicht vorhanden.
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, weil das Objekt bereits vorhanden ist.
500	Interner Fehler	Ein allgemeiner interner Fehler ist auf dem Server aufgetreten.
501	Nicht implementiert	Der URI ist bekannt, kann die Anforderung jedoch nicht ausführen.

Antwortkopfzeilen

In der vom Deploy-Server erzeugten HTTP-Antwort sind mehrere Header enthalten, darunter:

- Anforderungs-id
  Jeder erfolgreichen API-Anforderung wird eine eindeutige Anforderungskennung zugewiesen.
- Standort
  Wenn ein Objekt erstellt wird, enthält die Kopfzeile des Speicherorts die vollständige URL zum neuen Objekt einschließlich der eindeutigen Objektkennung.

Antwortkörper

Der Inhalt der mit einer API-Anfrage verknüpften Antwort ist je nach Objekt, Verarbeitungstyp und Erfolg oder Misserfolg der Anforderung unterschiedlich. Der Antwortkörper wird in JSON gerendert.

- Einzelnes Objekt
  Je nach Anforderung kann ein einzelnes Objekt mit einer Reihe von Feldern zurückgegeben werden. Beispielsweise können Sie GET verwenden, um ausgewählte Eigenschaften eines Clusters mit der

eindeutigen Kennung abzurufen.

- Mehrere Objekte
  Es können mehrere Objekte aus einer Ressourcensammlung zurückgegeben werden. In allen Fällen wird ein konsistentes Format verwendet, mit `num_records` Angabe der Anzahl der Datensätze und Datensätze, die ein Array der Objektinstanzen enthalten. Beispielsweise können Sie alle in einem bestimmten Cluster definierten Nodes abrufen.
- Jobobjekt
  Wenn ein API-Aufruf asynchron verarbeitet wird, wird ein Job-Objekt zurückgegeben, das den Hintergrund-Task ankers. Beispielsweise wird DIE POST-Anforderung, die zum Bereitstellen eines Clusters verwendet wird, asynchron bearbeitet und ein Job-Objekt zurückgegeben.
- Fehlerobjekt
  Wenn ein Fehler auftritt, wird immer ein Fehlerobjekt zurückgegeben. Beispielsweise erhalten Sie einen Fehler beim Versuch, ein Cluster mit einem bereits vorhandenen Namen zu erstellen.
- Leer
  In bestimmten Fällen werden keine Daten zurückgegeben und der Antworttext ist leer. Beispielsweise ist der Antwortkörper leer, nachdem Sie ZUM Löschen eines vorhandenen Hosts AUF „LÖSCHEN“ setzen.

Asynchrone Verarbeitung mit dem Job-Objekt

Einige der API-Aufrufe für die Bereitstellung, insbesondere solche, die eine Ressource erstellen oder ändern, können länger dauern als andere Anrufe. ONTAP Select implementieren verarbeitet diese langen Anforderungen asynchron.

Asynchrone Anforderungen, die mit Job Object beschrieben werden

Nach einem API-Aufruf, der asynchron ausgeführt wird, weist der HTTP-Antwortcode 202 darauf hin, dass die Anforderung erfolgreich validiert und akzeptiert, aber noch nicht abgeschlossen wurde. Die Anforderung wird als Hintergrundaufgabe verarbeitet, die nach der ersten HTTP-Antwort auf den Client weiter ausgeführt wird. Die Antwort umfasst das Job-Objekt, das die Anfrage einschließlich der eindeutigen Kennung anverankert.

Mithilfe der Seite ONTAP Select Deploy Online-Dokumentation können Sie ermitteln, welche API-Aufrufe asynchron funktionieren.

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

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

- Warteschlange
- Wird Ausgeführt
- Erfolg
- Ausfall

Es gibt zwei Verfahren, die Sie beim Abfragen eines Jobobjekts verwenden können, um einen Terminalstatus für die Aufgabe zu erkennen: Erfolg oder Fehler:

- Standard-Polling-Anfrage
  Der aktuelle Jobstatus wird sofort zurückgegeben

- Lange Abfrageanfrage
  Der Jobstatus wird nur zurückgegeben, wenn einer der folgenden Fehler auftritt:
  - Status hat sich vor kurzem geändert als der Datumswert, der auf der Abfrage angegeben wurde
  - Timeout-Wert abgelaufen (1 bis 120 Sekunden)

Standardabfrage und langes Abfragen verwenden denselben API-Aufruf, um ein Auftragsobjekt abzufragen. Eine lange Abfrageanforderung umfasst jedoch zwei Abfrageparameter: `poll_timeout` Und `last_modified`.

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

Allgemeines Verfahren für die Ausgabe einer asynchronen Anfrage

Sie können den folgenden grundlegenden Vorgang verwenden, um einen asynchronen API-Aufruf abzuschließen:

1. Geben Sie den asynchronen API-Aufruf aus.
2. Sie erhalten eine HTTP-Antwort 202, die darauf hinweist, dass die Anfrage erfolgreich angenommen wurde.
3. Extrahieren Sie die Kennung für das Job-Objekt aus dem Antwortkörper.
4. Führen Sie in einer Schleife in jedem Zyklus die folgenden Schritte aus:
   a. Den aktuellen Status des Jobs mit einer langen Umfrage abrufen
   b. Wenn sich der Job in einem nicht-Terminal-Status befindet (Warteschlange, wird ausgeführt), führen Sie die Schleife erneut aus.
5. Beenden Sie, wenn der Job einen Terminalstatus erreicht (Erfolg, Fehler).

Zugriff über einen Browser

Bevor Sie mit einem Browser auf die API zugreifen

Es gibt einige Dinge, die Sie beachten sollten, bevor Sie die Bereitstellung Online-Dokumentation Seite.

Implementierungsplan

Wenn Sie im Rahmen der Durchführung spezifischer Bereitstellungs- oder Verwaltungsaufgaben API-Aufrufe ausgeben möchten, sollten Sie die Erstellung eines Bereitstellungsplans in Betracht ziehen. Diese Pläne können formell oder informell sein und im Allgemeinen Ihre Ziele und die zu verwendenden API-Aufrufe enthalten. Weitere Informationen finden Sie unter Workflow-Prozesse mithilfe der REST-API implementieren.

JSON-Beispiele und Parameterdefinitionen

Jeder API-Aufruf wird auf der Dokumentationsseite in einem konsistenten Format beschrieben. Der Inhalt umfasst Implementierungsnotizen, Abfrageparameter und HTTP-Statuscodes. Außerdem können Sie wie folgt Details über den JSON anzeigen, der mit den API-Anfragen und Antworten verwendet wird:

- Beispielwert

Wenn Sie bei einem API-Aufruf auf *example Value* klicken, wird eine typische JSON-Struktur für den Aufruf angezeigt. Sie können das Beispiel je nach Bedarf ändern und als Eingabe für Ihre Anforderung verwenden.

- Modell
  Wenn Sie auf *Model* 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 mithilfe der Dokumentationsseite „Bereitstellen“ ausführen, sind Live-Vorgänge. Sie sollten darauf achten, dass Sie Konfigurationen oder andere Daten nicht versehentlich erstellen, aktualisieren oder löschen.

Rufen Sie die Seite Dokumentation bereitstellen auf

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

Bevor Sie beginnen

Sie müssen Folgendes haben:

- IP-Adresse oder Domain-Name der virtuellen ONTAP Select Deploy-Maschine
- 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. Melden Sie sich mit dem Benutzernamen und Passwort des Administrators an.

Ergebnis

Die Webseite Dokumentation bereitstellen wird angezeigt, auf der die Anrufe nach Kategorie unten auf der Seite organisiert sind.

API-Aufruf verstehen und ausführen

Die Details aller API-Aufrufe werden dokumentiert und in einem gemeinsamen Format auf der Webseite für die Online-Dokumentation von ONTAP Select Deploy angezeigt. Anhand eines einzelnen API-Anrufs können Sie auf die Details aller API-Aufrufe zugreifen und diese interpretieren.

Bevor Sie beginnen

Sie müssen auf der Webseite für die Online-Dokumentation von ONTAP Select Deploy angemeldet sein. Beim Erstellen des Clusters müssen Sie dem ONTAP Select-Cluster die eindeutige ID zuweisen.

Über diese Aufgabe

Sie können die Konfigurationsinformationen, die ein ONTAP Select Cluster beschreiben, mit seiner eindeutigen Kennung abrufen. In diesem Beispiel werden alle als kostengünstig klassifizierten Felder zurückgegeben. Als Best Practice sollten Sie jedoch nur die speziellen Felder anfordern, die erforderlich sind.

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 zur Rückgabe von Informationen über ein ONTAP Select-Cluster verwendet wird.

Workflow-Prozesse

Vor Verwendung der API-Workflows

Sie sollten sich auf die Überprüfung und Nutzung der Workflow-Prozesse vorbereiten.

Machen Sie sich mit den API-Aufrufen vertraut, die in den Workflows verwendet werden

Die Seite mit der ONTAP Select Online-Dokumentation enthält die Details zu jedem REST-API-Aufruf. Anstatt diese Details hier zu wiederholen, enthält jeder API-Aufruf, der in den Workflow-Samples verwendet wird, nur die Informationen, die Sie benötigen, um den Anruf auf der Dokumentationsseite zu finden. Nach dem Auffinden eines bestimmten API-Aufrufs können Sie die vollständigen Details des Anrufs überprüfen, einschließlich der Eingabeparameter, Ausgabeformate, HTTP-Statuscodes und des Aufruftyps.

Für jeden API-Aufruf in einem Workflow sind folgende Informationen enthalten, um den Anruf auf der Dokumentationsseite zu finden:

- Kategorie
  Die API-Aufrufe sind auf der Dokumentationsseite in funktional bezogene Bereiche oder Kategorien unterteilt. Um einen bestimmten API-Aufruf zu finden, blättern Sie zum unteren Seitenrand und klicken Sie auf die entsprechende API-Kategorie.
- HTTP-Verb
  Das HTTP-Verb identifiziert die Aktion, die für eine Ressource durchgeführt wird. Jeder API-Aufruf wird über ein einziges HTTP-Verb ausgeführt.
- Pfad
  Der Pfad bestimmt die spezifische Ressource, auf die die Aktion im Rahmen der Durchführung eines Anrufs zutrifft. Der Pfadstring wird an die Core-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 Seite mit der ONTAP Select-Dokumentation können Sie auch über eine Programmiersprache wie Python auf die REST-API zur Bereitstellung zugreifen. In diesem Fall unterscheidet sich die Core-URL etwas von der URL, die beim Zugriff auf die Seite mit der Online-Dokumentation verwendet wird. Wenn Sie direkt auf die API zugreifen, müssen Sie /API an die Domäne und den Port String anhängen. Beispiel: `http://deploy.mycompany.com/api`

Workflow 1: Erstellen eines Single-Node-Evaluierungsclusters auf ESXi

Sie können ein Single-Node ONTAP Select Cluster auf einem von vCenter verwalteten VMware ESXi Host implementieren. Der Cluster wird mit einer Evaluierungslizenz erstellt.

Der Workflow für die Cluster-Erstellung unterscheidet sich in folgenden Situationen:

- Der ESXi-Host wird nicht von vCenter gemanagt (Standalone-Host)

- Im Cluster werden mehrere Nodes oder Hosts verwendet
- Das Cluster wird in einer Produktionsumgebung mit einer erworbenen Lizenz implementiert
- Statt VMware ESXi wird der KVM-Hypervisor verwendet

1. Registrieren Sie die Anmeldedaten für vCenter-Server

Bei der Bereitstellung auf einem von einem vCenter-Server verwalteten ESXi-Host müssen Sie vor der Registrierung des Hosts eine Berechtigung hinzufügen. Das Deploy Administration Utility kann dann die Anmeldeinformationen zur Authentifizierung bei vCenter verwenden.

Kategorie	HTTP-Verb	Pfad
Implementieren	POST	/Sicherheit/Anmeldedaten

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"
}
```

Verarbeitungsart

Asynchron

Ausgabe

- Anmeldeinformations-ID in der Kopfzeile für Standortantwort
- Jobobjekt

2. Registrieren Sie einen Hypervisor-Host

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

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 (Schritt 02)

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

Verarbeitungsart

Asynchron

Ausgabe

- Host-ID in der Kopfzeile der Standortantwort
- Jobobjekt

3. Erstellen Sie einen Cluster

Wenn Sie ein ONTAP Select Cluster erstellen, wird die Basis-Cluster-Konfiguration registriert und die Node-Namen werden durch Deploy automatisch generiert.

Kategorie	HTTP-Verb	Pfad
Cluster	POST	/Cluster

Curl

Der Abfrageparameter Node_count sollte für ein Single-Node-Cluster 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-Eingang (Schritt 03)

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

Verarbeitungsart

Synchron

Ausgabe

- Cluster-ID in der Kopfzeile für Speicherantwort

4. Konfigurieren Sie den Cluster

Beim Konfigurieren des Clusters müssen Sie mehrere Attribute angeben.

Kategorie	HTTP-Verb	Pfad
Cluster	PATCH	/Clusters/{Cluster_id}

Curl

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-Eingang (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"}
}
```

Verarbeitungsart

Synchron

Ausgabe

Keine

5. Abrufen des Node-Namens

Das Deploy Administration Utility generiert automatisch die Node-IDs und Namen, wenn ein Cluster erstellt wird. Bevor Sie einen Node konfigurieren können, müssen Sie die zugewiesene ID abrufen.

Kategorie	HTTP-Verb	Pfad
Cluster	GET	/Clusters/{Cluster_id}/Nodes

Curl

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'
```

Verarbeitungsart

Synchron

Ausgabe

- Array zeichnet alle, die einen einzelnen Knoten mit der eindeutigen ID und dem Namen beschreiben

6. Konfigurieren Sie die Knoten

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

Kategorie	HTTP-Verb	Pfad
Cluster	PFAD	/Clusters/{Cluster_id}/Nodes/{Node_id}

Curl

Sie müssen 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)

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
}
```

Verarbeitungsart

Synchron

Ausgabe

Keine

7. Abrufen der Knoten-Netzwerke

Sie müssen die Daten und Managementnetzwerke identifizieren, die der Node im Single-Node-Cluster verwendet. Das interne Netzwerk wird nicht mit einem Single-Node-Cluster verwendet.

Kategorie	HTTP-Verb	Pfad
Cluster	GET	/Clusters/{Cluster_id}/Nodes/{Node_id}/Netzwerke

Curl

Sie müssen 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'
```

Verarbeitungsart

Synchron

Ausgabe

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

8. Konfigurieren Sie das Knoten Netzwerk

Sie müssen die Daten- und Managementnetzwerke konfigurieren. Das interne Netzwerk wird nicht mit einem Single-Node-Cluster verwendet.

Geben Sie den folgenden API-Aufruf zweimal ein, einmal für jedes Netzwerk.

Kategorie	HTTP-Verb	Pfad
Cluster	PATCH	/Clusters/{Cluster_id}/Nodes/{Node_id}/Networks/{Network_id}

Curl

Sie müssen die Cluster-ID, die Node-ID und die 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-Eingang (Schritt 08)

Sie müssen den Namen des Netzwerks angeben.

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

Verarbeitungsart

Synchron

Ausgabe

Keine

9. Konfigurieren Sie den Knoten Speicher-Pool

Der letzte Schritt beim Konfigurieren eines Node ist das Verbinden eines Speicherpools. Sie können die verfügbaren Speicherpools über den vSphere Web-Client oder optional über die Rest-API implementieren bestimmen.

Kategorie	HTTP-Verb	Pfad
Cluster	PATCH	/Clusters/{Cluster_id}/Nodes/{Node_id}/Networks/{Network_id}

Curl

Sie müssen die Cluster-ID, die Node-ID und die 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
    }
  ]
}
```

Verarbeitungsart

Synchron

Ausgabe

Keine

10. Implementieren Sie den Cluster

Nachdem das Cluster und der Node konfiguriert wurden, können Sie das Cluster implementieren.

Kategorie	HTTP-Verb	Pfad
Cluster	POST	/Clusters/{Cluster_id}/Deploy

Curl

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-Eingang (Schritt 10)

Sie müssen das Passwort für das ONTAP-Administratorkonto angeben.

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

Verarbeitungsart

Asynchron

Ausgabe

- Jobobjekt

Zugang mit Python

Bevor Sie mit Python auf die API zugreifen

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

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

- Die jeweils aktuelle Version von Python2 muss installiert sein.
  Die Beispielcodes wurden mit Python2 getestet. Sie sollten auch auf Python3 übertragbar sein, wurden aber nicht auf Kompatibilität getestet.
- Die Requests und urllib3-Bibliotheken müssen installiert sein.
  Sie können je nach Ihrer Umgebung Pip oder ein anderes Python Management Tool verwenden.
- Die Client-Workstation, auf der die Skripte ausgeführt werden, muss über einen Netzwerkzugriff auf die virtuelle ONTAP Select-Bereitstellungsmaschine verfügen.

Außerdem müssen Sie über die folgenden Informationen verfügen:

- IP-Adresse der virtuellen Maschine bereitstellen
- Benutzername und Kennwort eines Bereitstellungsadministrators

Verstehen Sie die Python-Skripte

Mit den Python-Beispielskripten können Sie mehrere verschiedene Aufgaben ausführen.

Sie sollten die Skripte verstehen, bevor Sie sie in einer Live-Deploy-Instanz verwenden.

Gemeinsame Designeigenschaften

Die Skripte wurden mit folgenden gemeinsamen Merkmalen entworfen:

- Ausführung über die Befehlszeilenschnittstelle auf einem Client-Rechner
  Sie können die Python-Skripte von jedem ordnungsgemäß konfigurierten Client-Computer aus ausführen. Weitere Informationen finden Sie unter *before you begin*.
- Akzeptieren Sie CLI-Eingabeparameter
  Jedes Skript wird über die CLI über Eingabeparameter gesteuert.
- Eingabedatei lesen
  Jedes Skript liest eine Eingabedatei basierend auf ihrem Zweck. Wenn Sie ein Cluster erstellen oder löschen, müssen Sie eine JSON-Konfigurationsdatei angeben. Beim Hinzufügen einer Node-Lizenz müssen Sie eine gültige Lizenzdatei angeben.
- Verwenden Sie ein gemeinsames Supportmodul
  Das gemeinsame Supportmodul *Deploy_Requests.py* enthält eine einzelne Klasse. Sie wird von jedem der Skripte importiert und verwendet.

Erstellen eines Clusters

Sie können mithilfe des Skripts Cluster.py einen ONTAP Select Cluster erstellen. Auf der Grundlage der CLI-Parameter und der Inhalte der JSON-Eingabedatei können Sie das Skript in Ihre Bereitstellungsumgebung wie folgt ändern:

- Hypervisor
  Sie können ESXI oder KVM bereitstellen (je nach Bereitstellungsversion). Bei der Implementierung in ESXi wird der Hypervisor durch vCenter gemanagt oder es kann ein Standalone-Host sein.
- Clustergröße
  Sie können ein Single-Node- oder Multi-Node-Cluster implementieren.
- 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
- Ausführliche Flag für Nachrichtenausgabe

Fügen Sie eine Node-Lizenz hinzu

Wenn Sie sich für die Bereitstellung eines Produktionsclusters entscheiden, müssen Sie für jeden Knoten eine Lizenz hinzufügen, indem Sie das Skript *add_license.py* verwenden. Sie können die Lizenz vor oder nach dem Implementieren 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
- Kennwort für den ONTAP-Benutzer

Löschen eines Clusters

Sie können einen vorhandenen ONTAP Select-Cluster mithilfe des Skripts *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 Clusters

Sie können mit dem folgenden Skript auf der Grundlage von Parametern, die in dem Skript definiert sind, und einer JSON-Eingabedatei einen Cluster 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 das Erstellen eines Clusters

Wenn Sie ein ONTAP Select-Cluster mithilfe der Python-Codebeispiele erstellen oder löschen, müssen Sie eine JSON-Datei als Eingabe für das Skript bereitstellen. Sie können das entsprechende JSON-Muster basierend auf Ihren Implementierungsplänen kopieren und ändern.

Single-Node-Cluster in 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
          }
        ]
      }
    }
  ]
}
```

Single-Node-Cluster in ESXi über 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
            }
          ]
        }
      }
    ]
}
```

Single-Node-Cluster 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 Node-Lizenz

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 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)
```

Common Support-Modul

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 zur Größenanpassung der Cluster-Nodes

Mit dem folgenden Skript können Sie die Größe der Nodes 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())
```