



Automatizza con RIPOSO

ONTAP Select

NetApp
February 03, 2026

Sommario

Automatizza con RIPOSO	1
Concetti	1
Base REST per i web Services	1
Come accedere all'API di implementazione	2
Implementare la versione delle API	2
Caratteristiche operative di base	3
Transazione API di richiesta e risposta	4
Elaborazione asincrona utilizzando l'oggetto lavoro	7
Accesso con un browser	8
Prima di accedere all'API con un browser	8
Accedere alla pagina della documentazione relativa alla distribuzione	9
Comprendere ed eseguire una chiamata API	9
Processi di workflow	10
Prima di utilizzare i flussi di lavoro API	10
Workflow 1: Creare un cluster di valutazione a nodo singolo su ESXi	11
Accesso con Python	17
Prima di accedere all'API utilizzando Python	17
Comprendere gli script Python	18
Esempi di codice Python	19
Script per creare un cluster	19
JSON per script per creare un cluster	26
Script per aggiungere una licenza del nodo	31
Script per eliminare un cluster	34
Modulo di supporto comune	36
Script per ridimensionare i nodi del cluster	40

Automatizza con RIPOSO

Concetti

Base REST per i web Services

Representational state Transfer (REST) è uno stile per la creazione di applicazioni web distribuite. Quando viene applicato alla progettazione di un'API di servizi Web, stabilisce un insieme di tecnologie e Best practice per esporre le risorse basate su server e gestirne gli stati. Utilizza protocolli e standard mainstream per fornire una base flessibile per l'implementazione e la gestione dei cluster ONTAP Select.

Architettura e limiti classici

REST fu formalmente articolata da Roy Fielding nel suo dottorato "[dissertazione](#)" presso la UC Irvine nel 2000. Definisce uno stile architettonico attraverso una serie di vincoli, che collettivamente hanno migliorato le applicazioni basate sul web e i protocolli sottostanti. I vincoli stabiliscono un'applicazione di servizi web RESTful basata su un'architettura client/server che utilizza un protocollo di comunicazione stateless.

Risorse e rappresentazione dello stato

Le risorse sono i componenti di base di un sistema basato su web. Quando si crea un'applicazione di servizi Web REST, le attività di progettazione iniziali includono:

- Identificazione delle risorse di sistema o basate su server che ogni sistema utilizza e gestisce le risorse. Una risorsa può essere un file, una transazione di business, un processo o un'entità amministrativa. Una delle prime attività nella progettazione di un'applicazione basata sui servizi web REST è quella di identificare le risorse.
- Definizione degli stati delle risorse e delle operazioni di stato associate le risorse si trovano sempre in un numero finito di stati. Gli stati, così come le operazioni associate utilizzate per influenzare i cambiamenti di stato, devono essere chiaramente definiti.

I messaggi vengono scambiati tra il client e il server per accedere e modificare lo stato delle risorse in base al modello generico CRUD (Create, Read, Update e Delete).

Endpoint URI

Ogni risorsa REST deve essere definita e resa disponibile utilizzando uno schema di indirizzamento ben definito. Gli endpoint in cui sono situate e identificate le risorse utilizzano un URI (Uniform Resource Identifier). L'URI fornisce un framework generale per la creazione di un nome univoco per ogni risorsa nella rete. L'URL (Uniform Resource Locator) è un tipo di URI utilizzato con i servizi Web per identificare e accedere alle risorse. Le risorse sono in genere esposte in una struttura gerarchica simile a una directory di file.

Messaggi HTTP

HTTP (Hypertext Transfer Protocol) è il protocollo utilizzato dal client e dal server dei servizi Web per scambiare messaggi di richiesta e risposta relativi alle risorse. Durante la progettazione di un'applicazione di servizi Web, i verbi HTTP (come GET e POST) vengono mappati alle risorse e alle azioni di gestione dello stato corrispondenti.

HTTP è stateless. Pertanto, per associare un insieme di richieste e risposte correlate in un'unica transazione,

è necessario includere informazioni aggiuntive nelle intestazioni HTTP portate con i flussi di dati di richiesta/risposta.

Formattazione JSON

Sebbene le informazioni possano essere strutturate e trasferite tra un client e un server in diversi modi, l'opzione più diffusa (e quella utilizzata con l'API DI DISTRIBUZIONE REST) è JavaScript Object Notation (JSON). JSON è uno standard di settore per la rappresentazione di semplici strutture di dati in testo normale e viene utilizzato per trasferire informazioni di stato che descrivono le risorse.

Come accedere all'API di implementazione

Grazie alla flessibilità intrinseca dei servizi web REST, è possibile accedere all'API di implementazione ONTAP Select in diversi modi.

Implementare l'interfaccia utente nativa dell'utility

Il modo principale per accedere all'API è tramite l'interfaccia utente Web di ONTAP Select Deploy. Il browser effettua chiamate all'API e riformatta i dati in base alla progettazione dell'interfaccia utente. È possibile accedere all'API anche tramite l'interfaccia della riga di comando dell'utilità di implementazione.

Pagina della documentazione online di ONTAP Select Deploy

La pagina della documentazione online di ONTAP Select Deploy fornisce un access point alternativo quando si utilizza un browser. Oltre a fornire un modo per eseguire direttamente le singole chiamate API, la pagina include anche una descrizione dettagliata dell'API, inclusi i parametri di input e altre opzioni per ciascuna chiamata. Le chiamate API sono organizzate in diverse aree funzionali o categorie.

Programma personalizzato

È possibile accedere all'API di distribuzione utilizzando uno dei diversi linguaggi e tool di programmazione. Le scelte più popolari includono Python, Java e CURL. Un programma, uno script o uno strumento che utilizza l'API agisce come un client di servizi Web REST. L'utilizzo di un linguaggio di programmazione consente di comprendere meglio l'API e offre l'opportunità di automatizzare le implementazioni di ONTAP Select.

Implementare la versione delle API

All'API REST inclusa nella distribuzione ONTAP Select viene assegnato un numero di versione. Il numero di versione dell'API è indipendente dal numero di release di implementazione. È necessario conoscere la versione dell'API inclusa nella release di deploy e il modo in cui questo potrebbe influire sull'utilizzo dell'API.

La versione corrente dell'utility di amministrazione di deploy include la versione 3 dell'API REST. Le versioni precedenti dell'utility di distribuzione includono le seguenti versioni API:

Implementare 2.8 e versioni successive

ONTAP Select Deploy 2.8 e tutte le versioni successive includono la versione 3 dell'API REST.

Implementare 2.7.2 e versioni precedenti

ONTAP Select Deploy 2.7.2 e tutte le release precedenti includono la versione 2 dell'API REST.



Le versioni 2 e 3 dell'API REST non sono compatibili. Se si esegue l'aggiornamento per la distribuzione della versione 2.8 o successiva da una release precedente che include la versione 2 dell'API, è necessario aggiornare qualsiasi codice esistente che acceda direttamente all'API e qualsiasi script che utilizzi l'interfaccia della riga di comando.

Caratteristiche operative di base

Mentre REST stabilisce un insieme comune di tecnologie e Best practice, i dettagli di ciascuna API possono variare in base alle scelte di progettazione. Prima di utilizzare l'API, è necessario conoscere i dettagli e le caratteristiche operative dell'API di implementazione di ONTAP Select.

Host hypervisor rispetto al nodo ONTAP Select

Un *host hypervisor* è la piattaforma hardware principale che ospita una macchina virtuale ONTAP Select. Quando una macchina virtuale ONTAP Select viene implementata e attiva su un host hypervisor, la macchina virtuale viene considerata un *nodo ONTAP Select*. Con la versione 3 dell'API REST di implementazione, gli oggetti host e nodo sono separati e distinti. Ciò consente una relazione uno-a-molti, in cui uno o più nodi ONTAP Select possono essere eseguiti sullo stesso host hypervisor.

Identificatori di oggetti

A ogni istanza o oggetto di risorsa viene assegnato un identificatore univoco al momento della creazione. Questi identificatori sono univoci a livello globale all'interno di una specifica istanza di ONTAP Select Deploy. Dopo aver emesso una chiamata API che crea una nuova istanza dell'oggetto, il valore id associato viene restituito al chiamante nell' `location` intestazione della risposta HTTP. È possibile estrarre l'identificatore e utilizzarlo nelle chiamate successive quando si fa riferimento all'istanza della risorsa.



Il contenuto e la struttura interna degli identificatori di oggetti possono cambiare in qualsiasi momento. È necessario utilizzare gli identificatori delle chiamate API applicabili solo se necessario quando si fa riferimento agli oggetti associati.

Richiedi identificatori

A ogni richiesta API riuscita viene assegnato un identificatore univoco. L'identificatore viene restituito nell' `request-id` intestazione della risposta HTTP associata. È possibile utilizzare un identificatore di richiesta per fare riferimento collettivamente alle attività di una singola transazione richiesta-risposta API specifica. Ad esempio, è possibile recuperare tutti i messaggi di evento per una transazione in base all'ID della richiesta.

Chiamate sincrone e asincrone

Esistono due modi principali in cui un server esegue una richiesta HTTP ricevuta da un client:

- Sincrono il server esegue immediatamente la richiesta e risponde con un codice di stato 200, 201 o 204.
- Asincrono il server accetta la richiesta e risponde con un codice di stato 202. Indica che il server ha accettato la richiesta del client e ha avviato un'attività in background per completare la richiesta. Il successo o l'errore finale non sono immediatamente disponibili e devono essere determinati tramite chiamate API aggiuntive.

Confermare il completamento di un lavoro a esecuzione prolungata

In genere, tutte le operazioni che possono richiedere molto tempo vengono elaborate in modo asincrono utilizzando un'attività in background sul server. Con l'API di distribuzione REST, ogni attività in background viene ancorata da un oggetto Job che tiene traccia dell'attività e fornisce informazioni, ad esempio lo stato corrente. Un oggetto Job, incluso il relativo identificatore univoco, viene restituito nella risposta HTTP dopo la creazione di un'attività in background.

È possibile eseguire query direttamente sull'oggetto Job per determinare il successo o l'errore della chiamata API associata. Per ulteriori informazioni, fare riferimento a *elaborazione asincrona mediante l'oggetto Job*.

Oltre all'utilizzo dell'oggetto Job, esistono altri modi per determinare il successo o il fallimento di una richiesta, tra cui:

- Messaggi di evento è possibile recuperare tutti i messaggi di evento associati a una chiamata API specifica utilizzando l'id della richiesta restituito con la risposta originale. I messaggi di evento contengono in genere un'indicazione di successo o di errore e possono essere utili anche quando si esegue il debug di una condizione di errore.
- Stato o stato delle risorse diverse risorse mantengono uno stato o un valore di stato che è possibile eseguire una query per determinare indirettamente il successo o l'errore di una richiesta.

Sicurezza

L'API di implementazione utilizza le seguenti tecnologie di sicurezza:

- Transport Layer Security tutto il traffico inviato in rete tra il server di implementazione e il client viene crittografato tramite TLS. L'utilizzo del protocollo HTTP su un canale non crittografato non è supportato. È supportata la versione TLS 1.2.
- Autenticazione HTTP l'autenticazione di base viene utilizzata per ogni transazione API. A ogni richiesta viene aggiunta un'intestazione HTTP, che include il nome utente e la password in una stringa base64.

Transazione API di richiesta e risposta

Ogni chiamata API di implementazione viene eseguita come richiesta HTTP alla macchina virtuale di implementazione, che genera una risposta associata al client. Questa coppia di richieste/risposte è considerata una transazione API. Prima di utilizzare l'API di distribuzione, è necessario conoscere le variabili di input disponibili per controllare una richiesta e il contenuto dell'output della risposta.

Variabili di input che controllano una richiesta API

È possibile controllare la modalità di elaborazione di una chiamata API tramite i parametri impostati nella richiesta HTTP.

Intestazioni delle richieste

È necessario includere diverse intestazioni nella richiesta HTTP, tra cui:

- Content-type se il corpo della richiesta include JSON, questa intestazione deve essere impostata su application/json.
- Accept (Accetta) se il corpo della risposta includerà JSON, questa intestazione deve essere impostata su Application/json.

- Autorizzazione l'autenticazione di base deve essere impostata con il nome utente e la password codificati in una stringa base64.

Corpo della richiesta

Il contenuto del corpo della richiesta varia in base alla chiamata specifica. Il corpo della richiesta HTTP è costituito da uno dei seguenti elementi:

- Oggetto JSON con variabili di input (ad esempio, il nome di un nuovo cluster)
- Vuoto

Filtra oggetti

Quando si esegue una chiamata API che utilizza GET, è possibile limitare o filtrare gli oggetti restituiti in base a qualsiasi attributo. Ad esempio, è possibile specificare un valore esatto da associare:

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

Oltre a una corrispondenza esatta, sono disponibili altri operatori per restituire un set di oggetti su un intervallo di valori. ONTAP Select supporta gli operatori di filtraggio indicati di seguito.

Operatore	Descrizione
=	Uguale a.
<	Inferiore a.
>	Maggiore di
≤	Minore o uguale a.
≥	Maggiore o uguale a.
	Oppure
!	Non uguale a.
*	Goloso carattere jolly

È inoltre possibile restituire un insieme di oggetti in base all'impostazione o meno di un campo specifico utilizzando la parola chiave Null o la relativa negazione (!null) come parte della query.

Selezione dei campi oggetto

Per impostazione predefinita, l'emissione di una chiamata API utilizzando GET restituisce solo gli attributi che identificano in modo univoco lo o gli oggetti. Questo insieme minimo di campi funge da chiave per ciascun oggetto e varia in base al tipo di oggetto. È possibile selezionare ulteriori proprietà dell'oggetto utilizzando il parametro di query dei campi nei seguenti modi:

- I campi economici specificano `fields=*` di recuperare i campi oggetto che sono mantenuti nella memoria del server locale o richiedono poca elaborazione per accedere.
- I campi costosi specificano `fields=**` di recuperare tutti i campi oggetto, compresi quelli che richiedono un'elaborazione server aggiuntiva per l'accesso.
- Selezione campo personalizzato utilizzare `fields=FIELDNAME` per specificare il campo esatto desiderato. Quando si richiedono più campi, i valori devono essere separati utilizzando virgole senza spazi.



Come Best practice, devi sempre identificare i campi specifici che desideri. È necessario recuperare l'insieme di campi economici o costosi solo quando necessario. La classificazione economica e costosa è determinata da NetApp in base all'analisi interna delle performance. La classificazione di un dato campo può cambiare in qualsiasi momento.

Ordinare gli oggetti nel set di output

I record di una raccolta di risorse vengono restituiti nell'ordine predefinito definito dall'oggetto. È possibile modificare l'ordine utilizzando il parametro query `Order_by` con il nome del campo e la direzione di ordinamento come segue:

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

Ad esempio, è possibile ordinare il campo tipo in ordine decrescente seguito da id in ordine crescente:

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

Quando si includono più parametri, è necessario separare i campi con una virgola.

Impaginazione

Quando si esegue una chiamata API utilizzando GET per accedere a un insieme di oggetti dello stesso tipo, vengono restituiti tutti gli oggetti corrispondenti per impostazione predefinita. Se necessario, è possibile limitare il numero di record restituiti utilizzando il parametro di query `max_records` con la richiesta. Ad esempio:

```
max_records=20
```

Se necessario, è possibile combinare questo parametro con altri parametri di query per restringere il set di risultati. Ad esempio, quanto segue restituisce fino a 10 eventi di sistema generati dopo l'ora specificata:

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

È possibile inviare più richieste per scorrere gli eventi (o qualsiasi tipo di oggetto). Ogni successiva chiamata API deve utilizzare un nuovo valore temporale basato sull'ultimo evento dell'ultimo set di risultati.

Interpretare una risposta API

Ogni richiesta API genera una risposta al client. È possibile esaminare la risposta per determinare se è stata eseguita correttamente e recuperare dati aggiuntivi in base alle necessità.

Codice di stato HTTP

I codici di stato HTTP utilizzati dall'API DI DISTRIBUZIONE REST sono descritti di seguito.

Codice	Significato	Descrizione
200	OK	Indica che le chiamate che non creano un nuovo oggetto sono riuscite.
201	Creato	Un oggetto è stato creato correttamente; l'intestazione della risposta di posizione include l'identificatore univoco dell'oggetto.
202	Accettato	Per eseguire la richiesta è stato avviato un processo in background a esecuzione prolungata, ma l'operazione non è ancora stata completata.
400	Richiesta errata	L'input della richiesta non viene riconosciuto o non è appropriato.
403	Vietato	Accesso negato a causa di un errore di autorizzazione.
404	Non trovato	La risorsa a cui si fa riferimento nella richiesta non esiste.

Codice	Significato	Descrizione
405	Metodo non consentito	Il verbo HTTP nella richiesta non è supportato per la risorsa.
409	Conflitto	Tentativo di creazione di un oggetto non riuscito perché l'oggetto esiste già.
500	Errore interno	Si è verificato un errore interno generale nel server.
501	Non implementato	L'URI è noto ma non è in grado di eseguire la richiesta.

Intestazioni delle risposte

Nella risposta HTTP generata dal server di implementazione sono incluse diverse intestazioni, tra cui:

- Request-id a ogni richiesta API riuscita viene assegnato un identificatore di richiesta univoco.
- Posizione quando viene creato un oggetto, l'intestazione di posizione include l'URL completo del nuovo oggetto, incluso l'identificatore univoco dell'oggetto.

Corpo di risposta

Il contenuto della risposta associata a una richiesta API varia in base all'oggetto, al tipo di elaborazione e all'esito positivo o negativo della richiesta. Il rendering del corpo di risposta viene eseguito in JSON.

- Oggetto singolo Un singolo oggetto può essere restituito con un insieme di campi in base alla richiesta. Ad esempio, È possibile utilizzare GET per recuperare le proprietà selezionate di un cluster utilizzando l'identificatore univoco.
- Oggetti multipli è possibile restituire più oggetti da una raccolta di risorse. In tutti i casi, viene utilizzato un formato coerente, con num_records l'indicazione del numero di record e record contenenti una matrice delle istanze dell'oggetto. Ad esempio, è possibile recuperare tutti i nodi definiti in un cluster specifico.
- Oggetto job se una chiamata API viene elaborata in modo asincrono, viene restituito un oggetto Job che ancora l'attività in background. Ad esempio, la richiesta POST utilizzata per implementare un cluster viene elaborata in modo asincrono e restituisce un oggetto Job.
- Oggetto Error se si verifica un errore, viene sempre restituito un oggetto Error. Ad esempio, quando si tenta di creare un cluster con un nome già esistente, viene visualizzato un messaggio di errore.
- Vuoto in alcuni casi, non viene restituito alcun dato e il corpo della risposta è vuoto. Ad esempio, il corpo della risposta è vuoto dopo aver utilizzato DELETE per eliminare un host esistente.

Elaborazione asincrona utilizzando l'oggetto lavoro

Alcune delle chiamate API di implementazione, in particolare quelle che creano o modificano una risorsa, possono richiedere più tempo per il completamento rispetto ad altre chiamate. ONTAP Select Deploy elabora queste richieste a esecuzione prolungata in modo asincrono.

Richieste asincrone descritte utilizzando l'oggetto Job

Dopo aver effettuato una chiamata API eseguita in modo asincrono, il codice di risposta HTTP 202 indica che la richiesta è stata convalidata e accettata correttamente, ma non ancora completata. La richiesta viene elaborata come attività in background che continua a essere eseguita dopo la risposta HTTP iniziale al client. La risposta include l'oggetto Job che ancora la richiesta, incluso il relativo identificatore univoco.



Fare riferimento alla pagina della documentazione online di ONTAP Select Deploy per determinare quali chiamate API funzionano in modo asincrono.

Eseguire una query sull'oggetto Job associato a una richiesta API

L'oggetto Job restituito nella risposta HTTP contiene diverse proprietà. È possibile eseguire una query sulla proprietà state per determinare se la richiesta è stata completata correttamente. Un oggetto Job può trovarsi in uno dei seguenti stati:

- In coda
- In esecuzione
- Successo
- Guasto

Esistono due tecniche che è possibile utilizzare quando si esegue il polling di un oggetto Job per rilevare lo stato di un terminale per l'attività, ovvero riuscito o non riuscito:

- Richiesta di polling standard lo stato corrente del processo viene restituito immediatamente
- Richiesta di polling lunga lo stato del processo viene restituito solo quando si verifica una delle seguenti condizioni:
 - Lo stato è stato modificato più di recente rispetto al valore data-ora fornito nella richiesta di polling
 - Il valore di timeout è scaduto (da 1 a 120 secondi)

Il polling standard e il polling lungo utilizzano la stessa chiamata API per eseguire query su un oggetto Job. Tuttavia, una richiesta di polling lunga include due parametri di query: `poll_timeout` E `last_modified`.



Per ridurre il carico di lavoro sulla macchina virtuale di implementazione, è necessario utilizzare sempre il polling lungo.

Procedura generale per l'emissione di una richiesta asincrona

È possibile utilizzare la seguente procedura di alto livello per completare una chiamata API asincrona:

1. Eseguire la chiamata API asincrona.
2. Ricevere una risposta HTTP 202 che indichi la corretta accettazione della richiesta.
3. Estrarre l'identificatore per l'oggetto Job dal corpo della risposta.
4. All'interno di un loop, eseguire le seguenti operazioni in ogni ciclo:
 - a. Ottenere lo stato corrente del lavoro con una richiesta di polling lungo
 - b. Se il job si trova in uno stato non terminale (in coda, in esecuzione), eseguire nuovamente il loop.
5. Interrompere quando il lavoro raggiunge uno stato terminale (successo, errore).

Accesso con un browser

Prima di accedere all'API con un browser

Prima di utilizzare la pagina di documentazione online di implementazione, è necessario

conoscere diversi aspetti.

Piano di implementazione

Se si intende effettuare chiamate API nell'ambito dell'esecuzione di attività amministrative o di implementazione specifiche, è consigliabile creare un piano di distribuzione. Questi piani possono essere formali o informali e generalmente contengono gli obiettivi e le chiamate API da utilizzare. Per ulteriori informazioni, fare riferimento a processi di workflow che utilizzano l'API DI distribuzione REST.

Esempi JSON e definizioni dei parametri

Ogni chiamata API viene descritta nella pagina della documentazione utilizzando un formato coerente. Il contenuto include note di implementazione, parametri di query e codici di stato HTTP. Inoltre, è possibile visualizzare i dettagli relativi al JSON utilizzato con le richieste e le risposte API come segue:

- Valore di esempio se si fa clic su *valore di esempio* in una chiamata API, viene visualizzata una struttura JSON tipica per la chiamata. È possibile modificare l'esempio in base alle esigenze e utilizzarlo come input per la richiesta.
- Modello facendo clic su *Model*, viene visualizzato un elenco completo dei parametri JSON, con una descrizione per ciascun parametro.

Prestare attenzione quando si emettono chiamate API

Tutte le operazioni API eseguite utilizzando la pagina della documentazione di implementazione sono operazioni live. Prestare attenzione a non creare, aggiornare o eliminare per errore la configurazione o altri dati.

Accedere alla pagina della documentazione relativa alla distribuzione

È necessario accedere alla pagina della documentazione online di ONTAP Select Deploy per visualizzare la documentazione API e per eseguire manualmente una chiamata API.

Prima di iniziare

È necessario disporre di quanto segue:

- Indirizzo IP o nome di dominio della macchina virtuale ONTAP Select Deploy
- Nome utente e password dell'amministratore

Fasi

1. Digitare l'URL nel browser e premere **Invio**:

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

2. Accedere utilizzando il nome utente e la password dell'amministratore.

Risultato

Viene visualizzata la pagina Web della documentazione di implementazione con le chiamate organizzate per categoria nella parte inferiore della pagina.

Comprendere ed eseguire una chiamata API

I dettagli di tutte le chiamate API vengono documentati e visualizzati in un formato

comune nella pagina Web della documentazione online di ONTAP Select Deploy. Conoscendo una singola chiamata API, è possibile accedere e interpretare i dettagli di tutte le chiamate API.

Prima di iniziare

È necessario accedere alla pagina Web della documentazione online di ONTAP Select Deploy. Al momento della creazione del cluster, è necessario disporre dell'identificatore univoco assegnato al cluster ONTAP Select.

A proposito di questa attività

È possibile recuperare le informazioni di configurazione che descrivono un cluster ONTAP Select utilizzando il relativo identificatore univoco. In questo esempio, vengono restituiti tutti i campi classificati come economici. Tuttavia, come Best practice, è necessario richiedere solo i campi specifici necessari.

Fasi

1. Nella pagina principale, scorrere fino in fondo e fare clic su **Cluster**.
2. Fare clic su **GET /clusters/{cluster_id}** per visualizzare i dettagli della chiamata API utilizzata per restituire informazioni su un cluster ONTAP Select.

Processi di workflow

Prima di utilizzare i flussi di lavoro API

È necessario prepararsi a rivedere e utilizzare i processi del workflow.

Comprendere le chiamate API utilizzate nei flussi di lavoro

La pagina della documentazione online di ONTAP Select include i dettagli di ogni chiamata REST API. Invece di ripetere questi dettagli qui, ogni chiamata API utilizzata negli esempi del flusso di lavoro include solo le informazioni necessarie per individuare la chiamata nella pagina della documentazione. Dopo aver individuato una chiamata API specifica, è possibile esaminare i dettagli completi della chiamata, inclusi i parametri di input, i formati di output, i codici di stato HTTP e il tipo di elaborazione della richiesta.

Le seguenti informazioni sono incluse per ogni chiamata API all'interno di un flusso di lavoro per facilitare l'individuazione della chiamata nella pagina della documentazione:

- Categorie le chiamate API sono organizzate nella pagina della documentazione in aree o categorie correlate alla funzionalità. Per individuare una chiamata API specifica, scorrere fino alla fine della pagina e fare clic sulla categoria API appropriata.
- Verbo HTTP il verbo HTTP identifica l'azione eseguita su una risorsa. Ogni chiamata API viene eseguita tramite un singolo verbo HTTP.
- Percorso il percorso determina la risorsa specifica a cui si applica l'azione durante l'esecuzione di una chiamata. La stringa del percorso viene aggiunta all'URL principale per formare l'URL completo che identifica la risorsa.

Creare un URL per accedere direttamente all'API REST

Oltre alla pagina di documentazione di ONTAP Select, è possibile accedere all'API DI DISTRIBUZIONE REST direttamente attraverso un linguaggio di programmazione come Python. In questo caso, l'URL principale è leggermente diverso dall'URL utilizzato per accedere alla pagina della documentazione online. Quando si accede direttamente all'API, è necessario aggiungere /api al dominio e alla stringa di porta. Ad esempio:

<http://deploy.mycompany.com/api>

Workflow 1: Creare un cluster di valutazione a nodo singolo su ESXi

È possibile implementare un cluster ONTAP Select a nodo singolo su un host VMware ESXi gestito da vCenter. Il cluster viene creato con una licenza di valutazione.

Il flusso di lavoro per la creazione del cluster è diverso nelle seguenti situazioni:

- L'host ESXi non è gestito da vCenter (host standalone)
- Nel cluster vengono utilizzati più nodi o host
- Il cluster viene implementato in un ambiente di produzione con una licenza acquistata
- L'hypervisor KVM viene utilizzato al posto di VMware ESXi

1. Registrare la credenziale del server vCenter

Durante la distribuzione su un host ESXi gestito da un server vCenter, è necessario aggiungere una credenziale prima di registrare l'host. L'utilità di amministrazione di deploy può quindi utilizzare la credenziale per l'autenticazione in vCenter.

Categoria	Verbo HTTP	Percorso
Implementare	POST	/security/credentials

Arricciatura

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

Ingresso JSON (step01)

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

Tipo di elaborazione

Asincrono

Output

- ID credenziale nell'intestazione della risposta di posizione
- Oggetto lavoro

2. Registrare un host hypervisor

È necessario aggiungere un host hypervisor in cui verrà eseguita la macchina virtuale contenente il nodo

ONTAP Select.

Categoria	Verbo HTTP	Percorso
Cluster	POST	/hosts

Arricciatura

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

Ingresso JSON (step02)

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

Tipo di elaborazione

Asincrono

Output

- ID host nell'intestazione di risposta della posizione
- Oggetto lavoro

3. Creare un cluster

Quando si crea un cluster ONTAP Select, la configurazione di base del cluster viene registrata e i nomi dei nodi vengono generati automaticamente da Deploy.

Categoria	Verbo HTTP	Percorso
Cluster	POST	/cluster

Arricciatura

Il parametro di query node_count deve essere impostato su 1 per un cluster a nodo singolo.

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

Ingresso JSON (step03)

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

Tipo di elaborazione

Sincrono

Output

- ID del cluster nell'intestazione della risposta di posizione

4. Configurare il cluster

Durante la configurazione del cluster, è necessario fornire diversi attributi.

Categoria	Verbo HTTP	Percorso
Cluster	PATCH	/cluster/{cluster_id}

Arricciatura

È necessario fornire l'ID del cluster.

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

Ingresso JSON (punto 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"}
}
```

Tipo di elaborazione

Sincrono

Output

Nessuno

5. Recuperare il nome del nodo

L'utility di amministrazione di deploy genera automaticamente gli identificatori e i nomi dei nodi quando viene creato un cluster. Prima di poter configurare un nodo, è necessario recuperare l'ID assegnato.

Categoria	Verbo HTTP	Percorso
Cluster	OTTIENI	/cluster/{cluster_id}/nodi

Arricciatura

È necessario fornire l'ID del cluster.

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

Tipo di elaborazione

Sincrono

Output

- Record di array ciascuno dei quali descrive un singolo nodo con ID e nome univoci

6. Configurare i nodi

È necessario fornire la configurazione di base per il nodo, che è la prima delle tre chiamate API utilizzate per configurare un nodo.

Categoria	Verbo HTTP	Percorso
Cluster	PERCORSO	/clusters/{cluster_id}/nodes/{node_id}

Arricciatura

È necessario fornire l'ID del cluster e l'ID del nodo.

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

Ingresso JSON (punto 06)

Specificare l'ID host in cui verrà eseguito il nodo ONTAP Select.

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

Tipo di elaborazione

Sincrono

Output

Nessuno

7. Recuperare le reti di nodi

È necessario identificare le reti di dati e di gestione utilizzate dal nodo nel cluster a nodo singolo. La rete interna non viene utilizzata con un cluster a nodo singolo.

Categoria	Verbo HTTP	Percorso
Cluster	OTTIENI	/clusters/{cluster_id}/nodes/{node_id}/networks

Arricciatura

È necessario fornire l'ID del cluster e l'ID del nodo.

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

Tipo di elaborazione

Sincrono

Output

- Array di due record ciascuno che descrive una singola rete per il nodo, inclusi ID e scopo univoci

8. Configurare la rete dei nodi

È necessario configurare le reti dati e di gestione. La rete interna non viene utilizzata con un cluster a nodo singolo.



Eseguire due volte la seguente chiamata API, una per ciascuna rete.

Categoria	Verbo HTTP	Percorso
Cluster	PATCH	/cluster/{cluster_id}/nodes/{node_id}/networks/{network_id}

Arricciatura

È necessario fornire l'ID del cluster, l'ID del nodo e l'ID di rete.

```
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'
```

Ingresso JSON (step08)

Specificare il nome della rete.

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

Tipo di elaborazione

Sincrono

Output

Nessuno

9. Configurare il pool di storage del nodo

La fase finale della configurazione di un nodo consiste nell'associare un pool di storage. È possibile determinare i pool di storage disponibili tramite il client Web vSphere o, facoltativamente, tramite l'API REST di implementazione.

Categoria	Verbo HTTP	Percorso
Cluster	PATCH	/cluster/{cluster_id}/nodes/{node_id}/networks/{network_id}

Arricciatura

È necessario fornire l'ID del cluster, l'ID del nodo e l'ID di rete.

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

Ingresso JSON (punto 09)

La capacità del pool è di 2 TB.

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

Tipo di elaborazione

Sincrono

Output

Nessuno

10. Implementare il cluster

Una volta configurati il cluster e il nodo, è possibile implementarlo.

Categoria	Verbo HTTP	Percorso
Cluster	POST	/clusters/{cluster_id}/deploy

Arricciatura

È necessario fornire l'ID del cluster.

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

Ingresso JSON (step10)

Specificare la password per l'account amministratore di ONTAP.

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

Tipo di elaborazione

Asincrono

Output

- Oggetto lavoro

Accesso con Python

Prima di accedere all'API utilizzando Python

È necessario preparare l'ambiente prima di eseguire gli script Python di esempio.

Prima di eseguire gli script Python, è necessario assicurarsi che l'ambiente sia configurato correttamente:

- È necessario installare l'ultima versione applicabile di python2. I codici campione sono stati testati utilizzando python2. Devono essere portatili anche per Python3, ma non sono stati testati per verificarne la compatibilità.
- Le richieste e le librerie urllib3 devono essere installate. È possibile utilizzare pip o un altro tool di gestione Python appropriato per il proprio ambiente.
- La workstation client in cui vengono eseguiti gli script deve disporre dell'accesso di rete alla macchina virtuale ONTAP Select Deploy.

Inoltre, è necessario disporre delle seguenti informazioni:

- Indirizzo IP della macchina virtuale di implementazione
- Nome utente e password di un account amministratore di implementazione

Comprendere gli script Python

Gli script Python di esempio consentono di eseguire diverse attività. È necessario comprendere gli script prima di utilizzarli in un'istanza di distribuzione live.

Caratteristiche di progettazione comuni

Gli script sono stati progettati con le seguenti caratteristiche comuni:

- Esecuzione dall'interfaccia della riga di comando su una macchina client è possibile eseguire gli script Python da qualsiasi macchina client correttamente configurata. Per ulteriori informazioni, consulta la sezione *before you begin*.
- Accettare i parametri di input CLI ogni script viene controllato in corrispondenza della CLI attraverso i parametri di input.
- Read input file ogni script legge un file di input in base al suo scopo. Quando si crea o si elimina un cluster, è necessario fornire un file di configurazione JSON. Quando si aggiunge una licenza nodo, è necessario fornire un file di licenza valido.
- Utilizza un modulo di supporto comune il modulo di supporto comune *deploy_requests.py* contiene una singola classe. Viene importato e utilizzato da ciascuno degli script.

Creare un cluster

È possibile creare un cluster ONTAP Select utilizzando lo script *cluster.py*. In base ai parametri CLI e al contenuto del file di input JSON, è possibile modificare lo script nell'ambiente di implementazione come segue:

- Hypervisor implementabili in ESXi o KVM (a seconda della release di implementazione). Durante l'implementazione in ESXi, l'hypervisor può essere gestito da vCenter o può essere un host standalone.
- Dimensione del cluster è possibile implementare un cluster a nodo singolo o a più nodi.
- Licenza di produzione o di valutazione è possibile implementare un cluster con una licenza di valutazione o acquistata per la produzione.

I parametri di input CLI per lo script includono:

- Nome host o indirizzo IP del server di implementazione
- Password per l'account utente admin
- Nome del file di configurazione JSON
- Flag dettagliato per l'output del messaggio

Aggiungere una licenza per nodi

Se si sceglie di implementare un cluster di produzione, è necessario aggiungere una licenza per ciascun nodo utilizzando lo script *add_License.py*. È possibile aggiungere la licenza prima o dopo l'implementazione del cluster.

I parametri di input CLI per lo script includono:

- Nome host o indirizzo IP del server di implementazione

- Password per l'account utente admin
- Nome del file di licenza
- Nome utente ONTAP con privilegi per aggiungere la licenza
- Password per l'utente ONTAP

Eliminare un cluster

È possibile eliminare un cluster ONTAP Select esistente utilizzando lo script *delete_cluster.py*.

I parametri di input CLI per lo script includono:

- Nome host o indirizzo IP del server di implementazione
- Password per l'account utente admin
- Nome del file di configurazione JSON

Esempi di codice Python

Script per creare un cluster

È possibile utilizzare lo script seguente per creare un cluster in base ai parametri definiti all'interno dello script e a un file di input JSON.

```
#!/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 presence 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 per script per creare un cluster

Quando si crea o si elimina un cluster ONTAP Select utilizzando gli esempi di codice Python, è necessario fornire un file JSON come input allo script. È possibile copiare e modificare l'esempio JSON appropriato in base ai piani di implementazione.

Cluster a nodo singolo su 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
}
]
}
}
]
}

```

Cluster a nodo singolo su ESXi con 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-Data",  
          "purpose": "storage",  
          "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
            }
        ]
    }
}
]
}

```

Cluster a nodo singolo su 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
}
]
}
}
]
```

```
    }
]
}
```

Script per aggiungere una licenza del nodo

È possibile utilizzare il seguente script per aggiungere una licenza per un nodo ONTAP Select.

```
#!/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 privilege 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)

```

Script per eliminare un cluster

È possibile utilizzare il seguente script CLI per eliminare un cluster esistente.

```

#!/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)

```

Modulo di supporto comune

Tutti gli script Python utilizzano una classe Python comune in un singolo modulo.

```

#!/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}

```

Script per ridimensionare i nodi del cluster

È possibile utilizzare lo script seguente per ridimensionare i nodi in un cluster ONTAP Select.

```

#!/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())

```

Informazioni sul copyright

Copyright © 2026 NetApp, Inc. Tutti i diritti riservati. Stampato negli Stati Uniti d'America. Nessuna porzione di questo documento soggetta a copyright può essere riprodotta in qualsiasi formato o mezzo (grafico, elettronico o meccanico, inclusi fotocopie, registrazione, nastri o storage in un sistema elettronico) senza previo consenso scritto da parte del detentore del copyright.

Il software derivato dal materiale sottoposto a copyright di NetApp è soggetto alla seguente licenza e dichiarazione di non responsabilità:

IL PRESENTE SOFTWARE VIENE FORNITO DA NETAPP "COSÌ COM'È" E SENZA QUALSIVOGLIA TIPO DI GARANZIA IMPLICITA O ESPRESSA FRA CUI, A TITOLO ESEMPLIFICATIVO E NON ESAUSTIVO, GARANZIE IMPLICITE DI COMMERCIALITÀ E IDONEITÀ PER UNO SCOPO SPECIFICO, CHE VENGONO DECLINATE DAL PRESENTE DOCUMENTO. NETAPP NON VERRÀ CONSIDERATA RESPONSABILE IN ALCUN CASO PER QUALSIVOGLIA DANNO DIRETTO, INDIRETTO, ACCIDENTALE, SPECIALE, ESEMPLARE E CONSEGUENZIALE (COMPRESI, A TITOLO ESEMPLIFICATIVO E NON ESAUSTIVO, PROCUREMENT O SOSTITUZIONE DI MERCI O SERVIZI, IMPOSSIBILITÀ DI UTILIZZO O PERDITA DI DATI O PROFITTI OPPURE INTERRUZIONE DELL'ATTIVITÀ AZIENDALE) CAUSATO IN QUALSIVOGLIA MODO O IN RELAZIONE A QUALUNQUE TEORIA DI RESPONSABILITÀ, SIA ESSA CONTRATTUALE, RIGOROSA O DOVUTA A INSOLVENZA (COMPRESA LA NEGLIGENZA O ALTRO) INSORTA IN QUALSIASI MODO ATTRAVERSO L'UTILIZZO DEL PRESENTE SOFTWARE ANCHE IN PRESENZA DI UN PREAVVISO CIRCA L'EVENTUALITÀ DI QUESTO TIPO DI DANNI.

NetApp si riserva il diritto di modificare in qualsiasi momento qualunque prodotto descritto nel presente documento senza fornire alcun preavviso. NetApp non si assume alcuna responsabilità circa l'utilizzo dei prodotti o materiali descritti nel presente documento, con l'eccezione di quanto concordato espressamente e per iscritto da NetApp. L'utilizzo o l'acquisto del presente prodotto non comporta il rilascio di una licenza nell'ambito di un qualche diritto di brevetto, marchio commerciale o altro diritto di proprietà intellettuale di NetApp.

Il prodotto descritto in questa guida può essere protetto da uno o più brevetti degli Stati Uniti, esteri o in attesa di approvazione.

LEGENDA PER I DIRITTI SOTTOPOSTI A LIMITAZIONE: l'utilizzo, la duplicazione o la divulgazione da parte degli enti governativi sono soggetti alle limitazioni indicate nel sottoparagrafo (b)(3) della clausola Rights in Technical Data and Computer Software del DFARS 252.227-7013 (FEB 2014) e FAR 52.227-19 (DIC 2007).

I dati contenuti nel presente documento riguardano un articolo commerciale (secondo la definizione data in FAR 2.101) e sono di proprietà di NetApp, Inc. Tutti i dati tecnici e il software NetApp forniti secondo i termini del presente Contratto sono articoli aventi natura commerciale, sviluppati con finanziamenti esclusivamente privati. Il governo statunitense ha una licenza irrevocabile limitata, non esclusiva, non trasferibile, non cedibile, mondiale, per l'utilizzo dei Dati esclusivamente in connessione con e a supporto di un contratto governativo statunitense in base al quale i Dati sono distribuiti. Con la sola esclusione di quanto indicato nel presente documento, i Dati non possono essere utilizzati, divulgati, riprodotti, modificati, visualizzati o mostrati senza la previa approvazione scritta di NetApp, Inc. I diritti di licenza del governo degli Stati Uniti per il Dipartimento della Difesa sono limitati ai diritti identificati nella clausola DFARS 252.227-7015(b) (FEB 2014).

Informazioni sul marchio commerciale

NETAPP, il logo NETAPP e i marchi elencati alla pagina <http://www.netapp.com/TM> sono marchi di NetApp, Inc. Gli altri nomi di aziende e prodotti potrebbero essere marchi dei rispettivi proprietari.