Automatizar con REST

ONTAP Select

NetApp
January 29, 2026

Tabla de contenidos

Automatizar con REST

Conceptos

Fundación de servicios web REST para implementar y administrar clústeres ONTAP Select

La Transferencia de Estado Representacional (REST) es un estilo para crear aplicaciones web distribuidas. Al aplicarla al diseño de una API de servicios web, establece un conjunto de tecnologías y mejores prácticas para exponer recursos basados en servidor y gestionar sus estados. Utiliza protocolos y estándares convencionales para proporcionar una base flexible para la implementación y gestión de clústeres de ONTAP Select .

Arquitectura y restricciones clásicas

REST fue articulado formalmente por Roy Fielding en su tesis doctoral. "disertación" En la Universidad de California en Irvine en el año 2000. Define un estilo arquitectónico mediante un conjunto de restricciones que, en conjunto, mejoran las aplicaciones web y los protocolos subyacentes. Las restricciones establecen una aplicación de servicios web RESTful basada en una arquitectura cliente-servidor que utiliza un protocolo de comunicación sin estado.

Recursos y representación estatal

Los recursos son los componentes básicos de un sistema web. Al crear una aplicación de servicios web REST, las primeras tareas de diseño incluyen:

- Identificación de recursos del sistema o del servidor. Todo sistema utiliza y mantiene recursos. Un recurso puede ser un archivo, una transacción comercial, un proceso o una entidad administrativa. Una de las primeras tareas al diseñar una aplicación basada en servicios web REST es identificar los recursos.
- Definición de los estados de los recursos y las operaciones de estado asociadas. Los recursos siempre se encuentran en uno de un número finito de estados. Los estados, así como las operaciones asociadas que se utilizan para modificarlos, deben estar claramente definidos.

Se intercambian mensajes entre el cliente y el servidor para acceder y cambiar el estado de los recursos de acuerdo con el modelo genérico CRUD (Crear, Leer, Actualizar y Eliminar).

Puntos finales URI

Cada recurso REST debe definirse y estar disponible mediante un esquema de direccionamiento bien definido. Los puntos finales donde se ubican e identifican los recursos utilizan un Identificador Uniforme de Recursos (URI). El URI proporciona un marco general para crear un nombre único para cada recurso en la red. El Localizador Uniforme de Recursos (URL) es un tipo de URI que se utiliza con servicios web para identificar y acceder a los recursos. Los recursos suelen exponerse en una estructura jerárquica similar a la de un directorio de archivos.

Mensajes HTTP

El Protocolo de Transferencia de Hipertexto (HTTP) es el protocolo que utilizan el cliente y el servidor de servicios web para intercambiar mensajes de solicitud y respuesta sobre los recursos. Al diseñar una aplicación de servicios web, los verbos HTTP (como GET y POST) se asignan a los recursos y a las acciones

de gestión de estado correspondientes.

HTTP no tiene estado. Por lo tanto, para asociar un conjunto de solicitudes y respuestas relacionadas en una misma transacción, se debe incluir información adicional en los encabezados HTTP que acompañan a los flujos de datos de solicitud/respuesta.

Formato JSON

Si bien la información se puede estructurar y transferir entre un cliente y un servidor de varias maneras, la opción más popular (y la que se utiliza con la API REST de Deploy) es la Notación de Objetos JavaScript (JSON). JSON es un estándar de la industria para representar estructuras de datos simples en texto plano y se utiliza para transferir información de estado que describe los recursos.

Cómo acceder a la API de implementación de ONTAP Select

Debido a la flexibilidad inherente de los servicios web REST, se puede acceder a la API de implementación de ONTAP Select de varias maneras diferentes.

Implementar la interfaz de usuario nativa de la utilidad

La principal forma de acceder a la API es a través de la interfaz web de ONTAP Select Deploy. El navegador realiza llamadas a la API y reformatea los datos según el diseño de la interfaz. También se accede a la API a través de la interfaz de línea de comandos de la utilidad Deploy.

Página de documentación en línea de ONTAP Select Deploy

La página de documentación en línea de ONTAP Select Deploy ofrece un punto de acceso alternativo al usar un navegador. Además de permitir la ejecución directa de llamadas a la API individuales, la página también incluye una descripción detallada de la API, incluyendo parámetros de entrada y otras opciones para cada llamada. Las llamadas a la API se organizan en varias áreas o categorías funcionales.

Programa personalizado

Puede acceder a la API de implementación mediante diversos lenguajes de programación y herramientas. Las opciones más populares incluyen Python, Java y cURL. Un programa, script o herramienta que utiliza la API actúa como un cliente de servicios web REST. Usar un lenguaje de programación le permite comprender mejor la API y le brinda la oportunidad de automatizar las implementaciones de ONTAP Select .

Control de versiones de la API de implementación de ONTAP Select

La API REST incluida con ONTAP Select Deploy tiene asignado un número de versión. Este número es independiente del número de versión de Deploy. Debe conocer la versión de la API incluida con su versión de Deploy y cómo esta podría afectar su uso de la API.

La versión actual de la utilidad de administración Deploy incluye la versión 3 de la API REST. Las versiones anteriores de la utilidad Deploy incluyen las siguientes versiones de la API:

Implementar 2.8 y posteriores

ONTAP Select Deploy 2.8 y todas las versiones posteriores incluyen la versión 3 de la API REST.

Implementar 2.7.2 y anteriores

ONTAP Select Deploy 2.7.2 y todas las versiones anteriores incluyen la versión 2 de la API REST.

> Las versiones 2 y 3 de la API REST no son compatibles. Si actualiza a Deploy 2.8 o posterior desde una versión anterior que incluye la versión 2 de la API, debe actualizar todo el código existente que acceda directamente a la API, así como cualquier script que utilice la interfaz de línea de comandos.

Características operativas básicas de la API de implementación de ONTAP Select

Si bien REST establece un conjunto común de tecnologías y mejores prácticas, los detalles de cada API pueden variar según las opciones de diseño. Debe conocer los detalles y las características operativas de la API ONTAP Select Deploy antes de usarla.

Host de hipervisor versus ONTAP Select

Un host de hipervisor es la plataforma de hardware principal que aloja una máquina virtual de ONTAP Select . Cuando una máquina virtual de ONTAP Select se implementa y está activa en un host de hipervisor, se considera un nodo de ONTAP Select. Con la versión 3 de la API REST de Deploy, los objetos de host y nodo son independientes. Esto permite una relación de uno a muchos, donde uno o más nodos de ONTAP Select pueden ejecutarse en el mismo host de hipervisor.

Identificadores de objetos

A cada instancia u objeto de recurso se le asigna un identificador único al crearse. Estos identificadores son únicos globalmente dentro de una instancia específica de ONTAP Select Deploy. Tras emitir una llamada a la API que crea una nueva instancia de objeto, el valor de ID asociado se devuelve al emisor en el… `location` Encabezado de la respuesta HTTP. Puede extraer el identificador y usarlo en llamadas posteriores al hacer referencia a la instancia del recurso.

> El contenido y la estructura interna de los identificadores de objeto pueden cambiar en cualquier momento. Solo debe usar los identificadores en las llamadas a la API correspondientes según sea necesario al referirse a los objetos asociados.

Identificadores de solicitud

A cada solicitud de API exitosa se le asigna un identificador único. El identificador se devuelve en el `request-id` Encabezado de la respuesta HTTP asociada. Puede usar un identificador de solicitud para referirse colectivamente a las actividades de una única transacción de solicitud-respuesta de API específica. Por ejemplo, puede recuperar todos los mensajes de evento de una transacción según el ID de la solicitud.

Llamadas sincrónicas y asincrónicas

Hay dos formas principales en que un servidor ejecuta una solicitud HTTP recibida de un cliente:

- Sincrónico El servidor ejecuta la solicitud inmediatamente y responde con un código de estado de 200, 201 o 204.
- Asíncrono: El servidor acepta la solicitud y responde con un código de estado 202. Esto indica que el servidor aceptó la solicitud del cliente e inició una tarea en segundo plano para completarla. El éxito o el fracaso final no está disponible de inmediato y debe determinarse mediante llamadas API adicionales.

Confirmar la finalización de un trabajo de larga duración

Generalmente, cualquier operación que tarde mucho tiempo en completarse se procesa asincrónicamente mediante una tarea en segundo plano en el servidor. Con la API REST de Deploy, cada tarea en segundo plano está anclada por un objeto Job que la rastrea y proporciona información, como su estado actual. Un objeto Job, con su identificador único, se devuelve en la respuesta HTTP tras crear una tarea en segundo plano.

Puede consultar directamente el objeto "Job" para determinar si la llamada a la API asociada se realizó correctamente o no. Consulte "Procesamiento asíncrono con el objeto "Job"" para obtener más información.

Además de utilizar el objeto Trabajo, existen otras formas de determinar el éxito o el fracaso de una solicitud, entre ellas:

- Mensajes de evento: Puede recuperar todos los mensajes de evento asociados a una llamada API específica utilizando el ID de solicitud devuelto con la respuesta original. Los mensajes de evento suelen indicar si la operación fue correcta o no, y también pueden ser útiles al depurar una condición de error.
- Estado o estado del recurso Varios de los recursos mantienen un valor de estado o estado que puede consultar para determinar indirectamente el éxito o el fracaso de una solicitud.

Seguridad

La API de implementación utiliza las siguientes tecnologías de seguridad:

- Seguridad de la capa de transporte: Todo el tráfico enviado a través de la red entre el servidor de implementación y el cliente se cifra mediante TLS. No se admite el uso del protocolo HTTP en un canal sin cifrar. Se admite la versión 1.2 de TLS.
- Autenticación HTTP. La autenticación básica se utiliza para cada transacción de la API. Se añade a cada solicitud un encabezado HTTP que incluye el nombre de usuario y la contraseña en una cadena base64.

Transacción de API de solicitud y respuesta para ONTAP Select

Cada llamada a la API de Deploy se realiza como una solicitud HTTP a la máquina virtual de Deploy, la cual genera una respuesta asociada para el cliente. Este par de solicitud/respuesta se considera una transacción de API. Antes de usar la API de Deploy, debe familiarizarse con las variables de entrada disponibles para controlar una solicitud y el contenido de la salida de la respuesta.

Variables de entrada que controlan una solicitud de API

Puede controlar cómo se procesa una llamada API a través de parámetros establecidos en la solicitud HTTP.

Encabezados de solicitud

Debe incluir varios encabezados en la solicitud HTTP, incluidos:

- content-type Si el cuerpo de la solicitud incluye JSON, este encabezado debe establecerse en application/json.
- aceptar Si el cuerpo de la respuesta incluirá JSON, este encabezado debe configurarse como application/json.
- La autenticación básica debe configurarse con el nombre de usuario y la contraseña codificados en una cadena base64.

Cuerpo de la solicitud

El contenido del cuerpo de la solicitud varía según la llamada específica. El cuerpo de la solicitud HTTP consta de uno de los siguientes elementos:

- Objeto JSON con variables de entrada (por ejemplo, el nombre de un nuevo clúster)
- Vacío

Filtrar objetos

Al ejecutar una llamada a la API que usa GET, puede limitar o filtrar los objetos devueltos según cualquier atributo. Por ejemplo, puede especificar un valor exacto para que coincida:

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

Además de la coincidencia exacta, existen otros operadores disponibles para devolver un conjunto de objetos en un rango de valores. ONTAP Select admite los operadores de filtrado que se muestran a continuación.

Operador	Descripción
=	Igual a
<	Menos que
>	Más que
⇐	Menor o igual a
>=	Mayor o igual que
|	O
!	No es igual a
*	Comodín codicioso

También puede devolver un conjunto de objetos en función de si un campo específico está configurado o no utilizando la palabra clave null o su negación (!null) como parte de la consulta.

Selección de campos de objeto

De forma predeterminada, al ejecutar una llamada a la API mediante GET, solo se devuelven los atributos que identifican de forma única el objeto o los objetos. Este conjunto mínimo de campos actúa como clave para cada objeto y varía según el tipo de objeto. Puede seleccionar propiedades adicionales del objeto mediante el parámetro de consulta "campos" de las siguientes maneras:

- Campos económicos Especificar `fields=*` para recuperar los campos de objeto que se mantienen en la memoria del servidor local o que requieren poco procesamiento para acceder.
- Campos costosos Especificar `fields=**` para recuperar todos los campos de objeto, incluidos aquellos que requieren procesamiento adicional del servidor para acceder.
- Selección de campo personalizado Usar `fields=FIELDNAME` Para especificar el campo exacto que desea. Al solicitar varios campos, los valores deben separarse con comas y sin espacios.

Como práctica recomendada, siempre debe identificar los campos específicos que desea. Solo debe recuperar el conjunto de campos económicos o costosos cuando sea necesario. NetApp determina la clasificación de económicos y costosos basándose en un análisis interno de rendimiento. La clasificación de un campo determinado puede cambiar en cualquier momento.

Ordenar objetos en el conjunto de salida

Los registros de una colección de recursos se devuelven en el orden predeterminado definido por el objeto. Puede cambiar el orden mediante el parámetro de consulta order_by con el nombre del campo y la dirección de ordenación, como se indica a continuación:

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

Por ejemplo, puede ordenar el campo de tipo en orden descendente seguido de id en orden ascendente:

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

Al incluir varios parámetros, debe separar los campos con una coma.

Paginación

Al ejecutar una llamada a la API mediante GET para acceder a una colección de objetos del mismo tipo, se devuelven todos los objetos coincidentes por defecto. Si es necesario, se puede limitar el número de registros devueltos mediante el parámetro de consulta max_records con la solicitud. Por ejemplo:

```
max_records=20
```

Si es necesario, puede combinar este parámetro con otros parámetros de consulta para limitar el conjunto de resultados. Por ejemplo, la siguiente operación devuelve hasta 10 eventos del sistema generados después del tiempo especificado:

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

Puedes emitir varias solicitudes para navegar por los eventos (o cualquier tipo de objeto). Cada llamada a la API posterior debe usar un nuevo valor de tiempo basado en el último evento del último conjunto de resultados.

Interpretar una respuesta de API

Cada solicitud de API genera una respuesta para el cliente. Puede examinarla para determinar si se realizó correctamente y recuperar datos adicionales según sea necesario.

Código de estado HTTP

A continuación se describen los códigos de estado HTTP utilizados por la API REST de implementación.

Código	Significado	Descripción
200	DE ACUERDO	Indica éxito para llamadas que no crean un nuevo objeto.
201	Creado	Se ha creado correctamente un objeto; el encabezado de respuesta de ubicación incluye el identificador único del objeto.
202	Aceptado	Se ha iniciado un trabajo en segundo plano de larga ejecución para ejecutar la solicitud, pero la operación aún no se ha completado.
400	Solicitud incorrecta	La entrada solicitada no se reconoce o es inadecuada.
403	Prohibido	Se deniega el acceso debido a un error de autorización.

Código	Significado	Descripción
404	Extraviado	El recurso al que se refiere la solicitud no existe.
405	Método no permitido	El verbo HTTP en la solicitud no es compatible con el recurso.
409	Conflicto	Se produjo un error al intentar crear un objeto porque el objeto ya existe.
500	Error interno	Se produjo un error interno general en el servidor.
501	No implementado	Se conoce el URI pero no es capaz de realizar la solicitud.

Encabezados de respuesta

Se incluyen varios encabezados en la respuesta HTTP generada por el servidor de implementación, incluidos:

- request-id A cada solicitud de API exitosa se le asigna un identificador de solicitud único.
- Ubicación Cuando se crea un objeto, el encabezado de ubicación incluye la URL completa del nuevo objeto, incluido el identificador de objeto único.

Cuerpo de la respuesta

El contenido de la respuesta asociada a una solicitud de API varía según el objeto, el tipo de procesamiento y si la solicitud se ha realizado correctamente o no. El cuerpo de la respuesta se representa en JSON.

- Objeto único. Se puede devolver un objeto único con un conjunto de campos según la solicitud. Por ejemplo, se puede usar GET para recuperar propiedades seleccionadas de un clúster mediante el identificador único.
- Múltiples objetos Se pueden devolver varios objetos de una colección de recursos. En todos los casos, se utiliza un formato consistente, con `num_records` Indica el número de registros y los registros que contienen una matriz de instancias de objeto. Por ejemplo, puede recuperar todos los nodos definidos en un clúster específico.
- Objeto de trabajo. Si una llamada a la API se procesa asincrónicamente, se devuelve un objeto de trabajo que ancla la tarea en segundo plano. Por ejemplo, la solicitud POST utilizada para implementar un clúster se procesa asincrónicamente y devuelve un objeto de trabajo.
- Objeto de error. Si se produce un error, siempre se devuelve un objeto de error. Por ejemplo, recibirá un error al intentar crear un clúster con un nombre ya existente.
- Vacío. En ciertos casos, no se devuelven datos y el cuerpo de la respuesta está vacío. Por ejemplo, el cuerpo de la respuesta está vacío después de usar DELETE para eliminar un host existente.

Procesamiento asincrónico mediante el objeto Job para ONTAP Select

Algunas llamadas a la API de Deploy, en particular las que crean o modifican un recurso, pueden tardar más en completarse que otras. ONTAP Select Deploy procesa estas solicitudes de larga duración de forma asincrónica.

Solicitudes asincrónicas descritas mediante el objeto Trabajo

Tras realizar una llamada a la API asincrónica, el código de respuesta HTTP 202 indica que la solicitud se ha validado y aceptado correctamente, pero aún no se ha completado. La solicitud se procesa como una tarea en segundo plano que continúa ejecutándose después de la respuesta HTTP inicial al cliente. La respuesta incluye el objeto Job que la ancla, incluyendo su identificador único.

Debe consultar la página de documentación en línea de ONTAP Select Deploy para determinar qué llamadas API funcionan de forma asincrónica.

Consultar el objeto de trabajo asociado con una solicitud de API

El objeto Job devuelto en la respuesta HTTP contiene varias propiedades. Puede consultar la propiedad de estado para determinar si la solicitud se completó correctamente. Un objeto Job puede estar en uno de los siguientes estados:

- En cola
- Correr
- Éxito
- Falla

Hay dos técnicas que puede utilizar al sondear un objeto de trabajo para detectar un estado terminal para la tarea, ya sea éxito o fracaso:

- Solicitud de sondeo estándar El estado actual del trabajo se devuelve inmediatamente
- Solicitud de sondeo larga El estado del trabajo se devuelve solo cuando ocurre una de las siguientes situaciones:
  - El estado ha cambiado más recientemente que el valor de fecha y hora proporcionado en la solicitud de encuesta
  - El valor de tiempo de espera ha expirado (de 1 a 120 segundos)

El sondeo estándar y el sondeo largo utilizan la misma llamada API para consultar un objeto de trabajo. Sin embargo, una solicitud de sondeo largo incluye dos parámetros de consulta: `poll_timeout` y `last_modified`.

Siempre debe utilizar un sondeo largo para reducir la carga de trabajo en la máquina virtual de implementación.

Procedimiento general para emitir una solicitud asincrónica

Puede utilizar el siguiente procedimiento de alto nivel para completar una llamada API asincrónica:

1. Emite la llamada API asincrónica.
2. Recibir una respuesta HTTP 202 indicando la aceptación exitosa de la solicitud.
3. Extraiga el identificador del objeto Trabajo del cuerpo de la respuesta.
4. Dentro de un bucle, realice lo siguiente en cada ciclo:
   a. Obtenga el estado actual del trabajo con una solicitud de sondeo largo
   b. Si el trabajo está en un estado no terminal (en cola, en ejecución), realice el bucle nuevamente.
5. Detenerse cuando el trabajo alcance un estado terminal (éxito, fracaso).

Acceder con un navegador

Antes de acceder a la API de implementación de ONTAP Select con un navegador

Hay varias cosas que debe tener en cuenta antes de utilizar la página de documentación en línea de Deploy.

Plan de implementación

Si planea realizar llamadas a la API al realizar tareas específicas de implementación o administración, considere crear un plan de implementación. Estos planes pueden ser formales o informales y generalmente contienen sus objetivos y las llamadas a la API que se utilizarán. Consulte Procesos de flujo de trabajo que utilizan la API REST de implementación para obtener más información.

Ejemplos JSON y definiciones de parámetros

Cada llamada a la API se describe en la página de documentación con un formato uniforme. El contenido incluye notas de implementación, parámetros de consulta y códigos de estado HTTP. Además, puede mostrar detalles sobre el JSON utilizado con las solicitudes y respuestas de la API, como se indica a continuación:

- Valor de ejemplo: Si hace clic en "Valor de ejemplo" en una llamada API, se mostrará una estructura JSON típica de la llamada. Puede modificar el ejemplo según sea necesario y usarlo como entrada para su solicitud.
- Modelo Si hace clic en *Modelo*, se muestra una lista completa de los parámetros JSON, con una descripción de cada parámetro.

Precaución al realizar llamadas API

Todas las operaciones de API que realice mediante la página de documentación de Implementación son operaciones en tiempo real. Tenga cuidado de no crear, actualizar ni eliminar por error la configuración ni otros datos.

Acceda a la página de documentación de ONTAP Select Deploy

Debe acceder a la página de documentación en línea de ONTAP Select Deploy para mostrar la documentación de la API, así como para emitir manualmente una llamada a la API.

Antes de empezar

Debes tener lo siguiente:

- Dirección IP o nombre de dominio de la máquina virtual ONTAP Select Deploy
- Nombre de usuario y contraseña del administrador

Pasos

1. Escribe la URL en tu navegador y pulsa **Enter**:

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

2. Sign in utilizando el nombre de usuario y la contraseña del administrador.

Resultado

La página web de documentación de Implementación se muestra con las llamadas organizadas por categoría en la parte inferior de la página.

Comprender y ejecutar una llamada API de ONTAP Select Deploy

Los detalles de todas las llamadas API se documentan y se muestran en un formato común en la página web de documentación en línea de ONTAP Select Deploy. Al comprender una sola llamada API, puede acceder e interpretar los detalles de todas las llamadas API.

Antes de empezar

Debe iniciar sesión en la página web de la documentación en línea de ONTAP Select Deploy. Debe tener el identificador único asignado a su clúster de ONTAP Select al crearlo.

Acerca de esta tarea

Puede recuperar la información de configuración que describe un clúster de ONTAP Select mediante su identificador único. En este ejemplo, se devuelven todos los campos clasificados como económicos. Sin embargo, se recomienda solicitar solo los campos específicos necesarios.

Pasos

1. En la página principal, desplácese hasta la parte inferior y haga clic en **Clúster**.
2. Haga clic en **GET /clusters/{cluster_id}** para mostrar los detalles de la llamada API utilizada para devolver información sobre un clúster de ONTAP Select .

Procesos de flujo de trabajo

Antes de utilizar los flujos de trabajo de la API de ONTAP Select

Debe prepararse para revisar y utilizar los procesos de flujo de trabajo.

Comprender las llamadas API utilizadas en los flujos de trabajo

La página de documentación en línea de ONTAP Select incluye los detalles de cada llamada a la API REST. En lugar de repetir estos detalles aquí, cada llamada a la API utilizada en los ejemplos de flujo de trabajo incluye solo la información necesaria para localizarla en la página de documentación. Después de localizar una llamada a la API específica, puede revisar todos sus detalles, incluyendo los parámetros de entrada, los formatos de salida, los códigos de estado HTTP y el tipo de procesamiento de la solicitud.

La siguiente información se incluye para cada llamada API dentro de un flujo de trabajo para ayudar a localizar la llamada en la página de documentación:

- Categoría: Las llamadas a la API se organizan en la página de documentación en áreas o categorías funcionalmente relacionadas. Para encontrar una llamada a la API específica, desplácese hasta la parte inferior de la página y haga clic en la categoría de API correspondiente.
- Verbo HTTP. El verbo HTTP identifica la acción realizada en un recurso. Cada llamada a la API se ejecuta mediante un único verbo HTTP.
- Ruta: La ruta determina el recurso específico al que se aplica la acción al realizar una llamada. La cadena de ruta se añade a la URL principal para formar la URL completa que identifica el recurso.

Construya una URL para acceder directamente a la API REST

Además de la página de documentación de ONTAP Select , también puede acceder a la API REST de Deploy directamente mediante un lenguaje de programación como Python. En este caso, la URL principal es

ligeramente diferente a la URL utilizada para acceder a la página de documentación en línea. Para acceder directamente a la API, debe añadir "/api" a la cadena de dominio y puerto. Por ejemplo: `http://deploy.mycompany.com/api`

Flujo de trabajo 1: Crear un clúster de evaluación de nodo único de ONTAP Select en ESXi

Puede implementar un clúster de ONTAP Select de un solo nodo en un host VMware ESXi administrado por vCenter. El clúster se crea con una licencia de evaluación.

El flujo de trabajo de creación de clúster difiere en las siguientes situaciones:

- El host ESXi no está administrado por vCenter (host independiente)
- Se utilizan varios nodos o hosts dentro del clúster
- El clúster se implementa en un entorno de producción con una licencia adquirida
- Se utiliza el hipervisor KVM en lugar de VMware ESXi

1. Registrar las credenciales del servidor vCenter

Al implementar en un host ESXi administrado por un servidor vCenter, debe agregar una credencial antes de registrar el host. La utilidad de administración de implementación puede usar la credencial para autenticarse en vCenter.

Categoría	verbo HTTP	Camino
Desplegar	CORREO	/seguridad/credenciales

Rizo

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

Entrada JSON (paso 01)

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

Tipo de procesamiento

Asincrónico

Producción

- ID de credencial en el encabezado de respuesta de ubicación
- Objeto de trabajo

2. Registrar un host de hipervisor

Debe agregar un host de hipervisor donde se ejecutará la máquina virtual que contiene el nodo ONTAP Select .

Categoría	verbo HTTP	Camino
Grupo	CORREO	/anfitriones

Rizo

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

Entrada JSON (paso 02)

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

Tipo de procesamiento

Asincrónico

Producción

- ID de host en el encabezado de respuesta de ubicación
- Objeto de trabajo

3. Crear un clúster

Cuando se crea un clúster de ONTAP Select , se registra la configuración básica del clúster y Deploy genera automáticamente los nombres de los nodos.

Categoría	verbo HTTP	Camino
Grupo	CORREO	/grupos

Rizo

El parámetro de consulta node_count debe establecerse en 1 para un clúster de un solo nodo.

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

Entrada JSON (paso 03)

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

Tipo de procesamiento

Sincrónico

Producción

- ID de clúster en el encabezado de respuesta de ubicación

4. Configurar el clúster

Hay varios atributos que debes proporcionar como parte de la configuración del clúster.

Categoría	verbo HTTP	Camino
Grupo	PARCHE	/clusters/{id_del_clúster}

Rizo

Debe proporcionar el ID del clúster.

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

Entrada JSON (paso 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 de procesamiento

Sincrónico

Producción

Ninguno

5. Recupere el nombre del nodo

La utilidad de administración de implementación genera automáticamente los identificadores y nombres de los nodos al crear un clúster. Antes de configurar un nodo, debe recuperar el ID asignado.

Categoría	verbo HTTP	Camino
Grupo	CONSEGUIR	/clusters/{id_de_cluster}/nodos

Rizo

Debe proporcionar el ID del clúster.

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

Tipo de procesamiento

Sincrónico

Producción

- La matriz registra cada uno describiendo un solo nodo con un ID y nombre únicos

6. Configurar los nodos

Debe proporcionar la configuración básica del nodo, que es la primera de las tres llamadas API utilizadas para configurar un nodo.

Categoría	verbo HTTP	Camino
Grupo	CAMINO	/clústeres/{id_de_clúster}/nodos/{id_de_nodo}

Rizo

Debe proporcionar el ID del clúster y el 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'
```

Entrada JSON (paso 06)

Debe proporcionar el ID del host donde se ejecutará el nodo ONTAP Select .

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

Tipo de procesamiento

Sincrónico

Producción

Ninguno

7. Recuperar las redes de nodos

Debe identificar las redes de datos y administración que utiliza el nodo en el clúster de un solo nodo. La red interna no se utiliza en un clúster de un solo nodo.

Categoría	verbo HTTP	Camino
Grupo	CONSEGUIR	/clústeres/{id_de_clúster}/nodos/{id_de_nodo}/redes

Rizo

Debe proporcionar el ID del clúster y el 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 de procesamiento

Sincrónico

Producción

- Matriz de dos registros, cada uno de los cuales describe una única red para el nodo, incluido el ID único y el propósito

8. Configurar la red del nodo

Debe configurar las redes de datos y administración. La red interna no se utiliza con un clúster de un solo nodo.

Emita la siguiente llamada API dos veces, una para cada red.

Categoría	verbo HTTP	Camino
Grupo	PARCHE	/clústeres/{id_de_clúster}/nodos/{id_de_nodo}/redes/{id_de_red}

Rizo

Debe proporcionar el ID del clúster, el ID del nodo y el ID de la red.

```
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'
```

Entrada JSON (paso 08)

Debe proporcionar el nombre de la red.

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

Tipo de procesamiento

Sincrónico

Producción

Ninguno

9. Configurar el grupo de almacenamiento del nodo

El último paso para configurar un nodo es conectar un pool de almacenamiento. Puede determinar los pools de almacenamiento disponibles mediante el cliente web de vSphere o, opcionalmente, mediante la API REST de implementación.

Categoría	verbo HTTP	Camino
Grupo	PARCHE	/clústeres/{id_de_clúster}/nodos/{id_de_nodo}/redes/{id_de_red}

Rizo

Debe proporcionar el ID del clúster, el ID del nodo y el ID de la red.

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

Entrada JSON (paso 09)

La capacidad del pool es de 2 TB.

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

Tipo de procesamiento

Sincrónico

Producción

Ninguno

10. Implementar el clúster

Una vez configurados el clúster y el nodo, puede implementar el clúster.

Categoría	verbo HTTP	Camino
Grupo	CORREO	/clusters/{id_de_cluster}/implementar

Rizo

Debe proporcionar el ID del clúster.

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

Entrada JSON (paso 10)

Debe proporcionar la contraseña para la cuenta de administrador de ONTAP .

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

Tipo de procesamiento

Asincrónico

Producción

- Objeto de trabajo

Información relacionada

"Implementar una instancia de evaluación de 90 días de un clúster ONTAP Select"

Acceso con Python

Antes de acceder a la API de implementación de ONTAP Select mediante Python

Debe preparar el entorno antes de ejecutar los scripts de Python de muestra.

Antes de ejecutar los scripts de Python, debe asegurarse de que el entorno esté configurado correctamente:

- Se requiere la última versión aplicable de Python2. Los códigos de ejemplo se han probado con Python2. Deberían ser portables a Python3, pero no se ha probado su compatibilidad.
- Las bibliotecas Requests y urllib3 deben estar instaladas. Puede usar pip u otra herramienta de administración de Python según sea necesario para su entorno.

- La estación de trabajo cliente donde se ejecutan los scripts debe tener acceso de red a la máquina virtual ONTAP Select Deploy.

Además, deberás contar con la siguiente información:

- Dirección IP de la máquina virtual de implementación
- Nombre de usuario y contraseña de una cuenta de administrador de Deploy

Comprenda los scripts de Python para ONTAP Select Deploy

Los scripts de ejemplo de Python permiten realizar diversas tareas. Es recomendable comprenderlos antes de usarlos en una instancia de Deploy activa.

Características de diseño comunes

Los scripts han sido diseñados con las siguientes características comunes:

- Ejecutar desde la interfaz de línea de comandos en un equipo cliente. Puede ejecutar los scripts de Python desde cualquier equipo cliente correctamente configurado. Consulte "Antes de comenzar" para obtener más información.
- Aceptar parámetros de entrada CLI Cada script se controla en la CLI a través de parámetros de entrada.
- Leer archivo de entrada. Cada script lee un archivo de entrada según su propósito. Al crear o eliminar un clúster, debe proporcionar un archivo de configuración JSON. Al agregar una licencia de nodo, debe proporcionar un archivo de licencia válido.
- Usar un módulo de soporte común. El módulo de soporte común *deploy_requests.py* contiene una sola clase. Se importa y utiliza en cada script.

Crear un clúster

Puede crear un clúster de ONTAP Select con el script cluster.py. Según los parámetros de la CLI y el contenido del archivo de entrada JSON, puede adaptar el script a su entorno de implementación de la siguiente manera:

- Hipervisor: Puede implementar en ESXI o KVM (según la versión de implementación). Al implementar en ESXi, el hipervisor puede ser administrado por vCenter o puede ser un host independiente.
- Tamaño del clúster Puede implementar un clúster de un solo nodo o de varios nodos.
- Licencia de evaluación o producción Puede implementar un clúster con una licencia de evaluación o comprada para producción.

Los parámetros de entrada CLI para el script incluyen:

- Nombre de host o dirección IP del servidor de implementación
- Contraseña para la cuenta de usuario administrador
- Nombre del archivo de configuración JSON
- Bandera verbosa para la salida del mensaje

Agregar una licencia de nodo

Si decide implementar un clúster de producción, debe agregar una licencia para cada nodo mediante el script *add_license.py*. Puede agregar la licencia antes o después de implementar el clúster.

Los parámetros de entrada CLI para el script incluyen:

- Nombre de host o dirección IP del servidor de implementación
- Contraseña para la cuenta de usuario administrador
- Nombre del archivo de licencia
- Nombre de usuario de ONTAP con privilegios para agregar la licencia
- Contraseña para el usuario de ONTAP

Eliminar un clúster

Puede eliminar un clúster de ONTAP Select existente mediante el script *delete_cluster.py*.

Los parámetros de entrada CLI para el script incluyen:

- Nombre de host o dirección IP del servidor de implementación
- Contraseña para la cuenta de usuario administrador
- Nombre del archivo de configuración JSON

Ejemplos de código de Python

Script para crear un clúster de ONTAP Select

Puede utilizar el siguiente script para crear un clúster basado en parámetros definidos dentro del script y un archivo de entrada 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 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 para script para crear un clúster de ONTAP Select

Al crear o eliminar un clúster de ONTAP Select con ejemplos de código de Python, debe proporcionar un archivo JSON como entrada al script. Puede copiar y modificar el ejemplo JSON correspondiente según sus planes de implementación.

Clúster de un solo nodo en 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
          }
        ]
      }
    }
```

```
  ]
}
```

Clúster de un solo nodo en ESXi mediante 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
            }
          ]
        }
      }
    ]
}
```

Clúster de un solo nodo en 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 para agregar una licencia de nodo ONTAP Select

Puede utilizar el siguiente script para agregar una licencia para 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 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)
```

Script para eliminar un clúster de ONTAP Select

Puede utilizar el siguiente script CLI para eliminar un clúster existente.

```
#!/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)
```

Módulo de Python de soporte común para ONTAP Select

Todos los scripts de Python utilizan una clase Python común en un solo módulo.

```
#!/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 para cambiar el tamaño de los nodos del clúster ONTAP Select

Puede utilizar el siguiente script para cambiar el tamaño de los nodos en un clúster de 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())
```