Automatiza con REST

ONTAP Select

NetApp
November 08, 2026

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

Tabla de contenidos

Automatiza con REST . . . 1
- Conceptos . . . 1
  - Infórmate sobre los servicios web REST para implementar y gestionar clústeres de ONTAP Select . . . 1
  - Infórmate sobre cómo acceder a la API de ONTAP Select Deploy . . . 2
  - Características operativas básicas de la API de ONTAP Select Deploy . . . 2
  - Transacción de API de solicitud y respuesta para ONTAP Select . . . 4
  - Procesamiento asíncrono mediante el objeto Job para ONTAP Select . . . 7
- Accede con un navegador . . . 8
  - Descubre cómo acceder a la API de ONTAP Select Deploy desde un navegador . . . 8
  - Accede a la página de documentación de ONTAP Select Deploy . . . 9
  - Comprende y ejecuta una llamada a la API de ONTAP Select Deploy . . . 9
- Procesos de flujo de trabajo . . . 10
  - Infórmate sobre cómo usar los flujos de trabajo de la API de ONTAP Select Deploy . . . 10
  - Crear un clúster de evaluación de ONTAP Select de un solo nodo en ESXi . . . 10
- Acceso con Python . . . 17
  - Descubre cómo acceder a la API de ONTAP Select Deploy mediante Python . . . 17
  - Infórmate sobre los scripts de Python para ONTAP Select Deploy . . . 17
- Ejemplos de código Python . . . 19
  - Script para crear un clúster ONTAP Select . . . 19
  - JSON para el script para crear un clúster ONTAP Select . . . 26
  - Script para añadir una licencia de nodo ONTAP Select . . . 30
  - Script para eliminar un clúster de ONTAP Select . . . 34
  - Módulo Python de soporte común para ONTAP Select . . . 36
  - Script para redimensionar los nodos del clúster ONTAP Select . . . 40

Automatiza con REST

Conceptos

Infórmate sobre los servicios web REST para implementar y gestionar clústeres de ONTAP Select

Representational State Transfer (REST) es un estilo para crear aplicaciones web distribuidas. Al aplicarse al diseño de una API de servicios web, establece un conjunto de tecnologías y buenas prácticas para exponer recursos basados en servidores y gestionar sus estados. Utiliza protocolos y estándares convencionales para proporcionar una base flexible para la implementación y gestión de clústeres ONTAP Select.

Arquitectura y limitaciones clásicas

REST fue formalmente descrito por Roy Fielding en su tesis doctoral "disertación" en UC Irvine en 2000. Define un estilo arquitectónico mediante un conjunto de restricciones, que en conjunto han mejorado 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 de estado

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

- Identificación de recursos del sistema o del servidor Cada 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 estados de recursos y operaciones de estado asociadas Los recursos siempre se encuentran en uno de un número finito de estados. Tanto los estados como las operaciones asociadas que modifican dichos estados deben estar claramente definidos.

Se intercambian mensajes entre el cliente y el servidor para acceder a los recursos y modificar su estado según el modelo CRUD (Crear, Leer, Actualizar y Eliminar) genérico.

Puntos finales URI

Cada recurso REST debe definirse y estar disponible mediante un esquema de direccionamiento bien definido. Los endpoints 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 Uniform Resource Locator (URL) es un tipo de URI que se utiliza con los servicios web para identificar y acceder a los recursos. Los recursos suelen exponerse en una estructura jerárquica similar a un directorio de archivos.

Mensajes de 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. Como parte del diseño de 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 mantiene estado. Por lo tanto, para asociar un conjunto de solicitudes y respuestas relacionadas bajo 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 diversas maneras, la opción más popular (y la que se utiliza con la API de REST de Deploy) es JavaScript Object Notation (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.

Infórmate sobre cómo acceder a la API de ONTAP Select Deploy

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

> La API de REST incluida con ONTAP Select Deploy tiene asignado un número de versión. El número de versión de la API es independiente del número de versión de Deploy. La utilidad de administración de ONTAP Select 9.19.1 Deploy incluye la versión 3 de la API de REST.

Implementa la interfaz de usuario nativa de la utilidad

La principal forma de acceder a la API es a través de la interfaz web de usuario de ONTAP Select Deploy. El navegador realiza llamadas a la API y reformatea los datos según el diseño de la interfaz de usuario. También accedes a la API mediante 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 ejecutar llamadas a la API directamente, la página también incluye una descripción detallada de la API, incluidos los parámetros de entrada y otras opciones para cada llamada. Las llamadas a la API están organizadas en varias áreas o categorías funcionales diferentes.

Programa personalizado

Puedes acceder a la API de Deploy utilizando diversos lenguajes y herramientas de programación. Entre las opciones más populares se encuentran Python, Java y cURL. Un programa, script o herramienta que utilice la API actúa como cliente de servicios web REST. El uso de un lenguaje de programación te permite comprender mejor la API y te da la oportunidad de automatizar las implementaciones de ONTAP Select.

Características operativas básicas de la API de ONTAP Select Deploy

Si bien REST establece un conjunto común de tecnologías y mejores prácticas, los detalles de cada API pueden variar según las decisiones de diseño. Debes familiarizarte con los detalles y las características operativas de la API de ONTAP Select Deploy antes de usar la API.

Host del hipervisor frente a nodo ONTAP Select

Un *host de hipervisor* es la plataforma de hardware central que aloja una máquina virtual ONTAP Select. Cuando una máquina virtual ONTAP Select se implementa y está activa en un host hipervisor, la máquina virtual se considera un nodo ONTAP Select. Con la versión 3 de la API de REST Deploy, los objetos host y

nodo son independientes y distintos. Esto permite una relación de uno a muchos, donde uno o más nodos ONTAP Select pueden ejecutarse en el mismo host hipervisor.

Identificadores de objetos

Cada instancia u objeto de recurso recibe un identificador único al crearse. Estos identificadores son únicos a nivel global dentro de una instancia específica de ONTAP Select Deploy. Tras realizar una llamada a la API que crea una nueva instancia de objeto, el valor del identificador asociado se devuelve al emisor en el `location` encabezado de la respuesta HTTP. Puedes extraer el identificador y usarlo en llamadas posteriores cuando te refieras a la instancia del recurso.



El contenido y la estructura interna de los identificadores de objeto pueden cambiar en cualquier momento. Debes utilizar los identificadores únicamente en las llamadas a la API de REST correspondientes, cuando sea necesario para referirte a los objetos asociados.

Identificadores de solicitud

Cada solicitud API exitosa recibe un identificador único. El identificador se devuelve en el `request-id` header de la respuesta HTTP asociada. Puedes usar un identificador de solicitud para referirte colectivamente a las actividades de una transacción API de solicitud-respuesta específica. Por ejemplo, puedes recuperar todos los mensajes de evento de una transacción basándote en el request id.

Llamadas síncronas y asíncronas

Existen dos formas principales en que un servidor procesa una solicitud HTTP recibida de un cliente:

- Síncrono El servidor realiza la solicitud inmediatamente y responde con un código de estado 200, 201 o 204.
- Asíncrono. El servidor acepta la solicitud y responde con un código de estado 202. Esto indica que el servidor ha aceptado la solicitud del cliente y ha iniciado una tarea en segundo plano para completarla. El resultado final (éxito o fracaso) no está disponible de inmediato y debe determinarse mediante llamadas adicionales a la API de REST.

Confirma la finalización de un trabajo de larga duración

Por lo general, cualquier operación que pueda tardar mucho tiempo en completarse se procesa de forma asíncrona mediante una tarea en segundo plano en el servidor. Con la API de REST de Deploy, cada tarea en segundo plano está vinculada a un objeto Job que realiza un seguimiento de la tarea y proporciona información, como el estado actual. Un objeto Job, incluido su identificador único, se devuelve en la respuesta HTTP después de que se crea una tarea en segundo plano.

Puedes consultar directamente el objeto Job para determinar el éxito o el fallo de la llamada a la API asociada. Consulta *procesamiento asíncrono mediante el objeto Job* para obtener más información.

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

- Mensajes de evento Puedes recuperar todos los mensajes de evento asociados a una llamada específica a la API utilizando el id de solicitud que se devuelve con la respuesta original. Los mensajes de evento suelen contener una indicación de éxito o fallo, y también pueden ser útiles al depurar una condición de error.
- Estado o condición del recurso Varios de los recursos mantienen un valor de estado o condición que puedes consultar para determinar indirectamente el éxito o el fracaso de una solicitud.

Seguridad

La API de Deploy 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 Deploy y el cliente está cifrado mediante TLS. No se admite el uso del protocolo HTTP a través de un canal no cifrado. Se admite la versión 1.2 de TLS.
- La autenticación HTTP básica se utiliza para cada transacción de API. A cada solicitud se le agrega 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 solicitud/respuesta se considera una transacción de API. Antes de usar la API de Deploy, debes familiarizarte 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

Puedes controlar cómo se procesa una llamada a la API mediante parámetros establecidos en la solicitud HTTP.

Encabezados de la solicitud

Debes incluir varios encabezados en la solicitud HTTP, entre ellos:

- content-type Si el cuerpo de la solicitud incluye JSON, este encabezado debe establecerse en application/json.
- accept Si el cuerpo de la respuesta incluye JSON, este encabezado debe establecerse en 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 (como, por ejemplo, el nombre de un nuevo clúster)
- Vacío

Filtrar objetos

Al realizar una llamada a la API que utiliza GET, puedes limitar o filtrar los objetos devueltos en función de cualquier atributo. Por ejemplo, puedes 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 dentro de 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 que
>=	Mayor o igual que
	O
!	No es igual a
*	Comodín codicioso

También puedes devolver un conjunto de objetos en función de si un campo específico está definido o no, usando la palabra clave null o su negación (!null) como parte de la consulta.

Selección de campos de objeto

Por defecto, al realizar 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. Puedes seleccionar propiedades adicionales del objeto usando el parámetro de consulta fields de las siguientes maneras:

- Campos económicos Especifica `fields=*` para recuperar los campos del objeto que se mantienen en la memoria del servidor local o que requieren poco procesamiento para acceder a ellos.
- Campos costosos Especifica `fields=**` para recuperar todos los campos del objeto, incluidos aquellos que requieren procesamiento adicional del servidor para acceder a ellos.
- Selección de campo personalizado Usa `fields=FIELDNAME` para especificar el campo exacto que quieres. Al solicitar varios campos, los valores deben estar separados por comas sin espacios.

> Como buena práctica, siempre debes identificar los campos específicos que quieres. Solo debes recuperar el conjunto de campos económicos o costosos cuando lo necesites. La clasificación de campos económicos y costosos la determina NetApp según un análisis interno de rendimiento. La clasificación de un campo determinado puede cambiar en cualquier momento.

Ordena los objetos en el conjunto de salida

Los registros de una colección de recursos se devuelven en el orden predeterminado definido por el objeto. Puedes cambiar el orden utilizando el parámetro de consulta order_by con el nombre del campo y la dirección de ordenación de la siguiente manera:

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

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

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

Cuando incluyas varios parámetros, debes separar los campos con una coma.

Paginación

Al realizar una llamada a la API mediante GET para acceder a una colección de objetos del mismo tipo, se devuelven por defecto todos los objetos coincidentes. Si lo necesitas, puedes limitar el número de registros devueltos usando el parámetro de consulta max_records en la solicitud. Por ejemplo:
`max_records=20`

Si es necesario, puedes combinar este parámetro con otros parámetros de consulta para acotar el conjunto de resultados. Por ejemplo, la siguiente consulta devuelve hasta 10 eventos del sistema generados después de la hora especificada:
`time⇒ 2019-04-04T15:41:29.140265Z&max_records=10`

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

Interpretar una respuesta de API

Cada solicitud a la API genera una respuesta que se envía al cliente. Puedes examinar la respuesta para determinar si fue exitosa y recuperar datos adicionales según lo necesites.

Código de estado HTTP

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

Código	Significado	Descripción
200	OK	Indica el éxito para las llamadas que no crean un nuevo objeto.
201	Creado	Se ha creado un objeto correctamente; el encabezado de respuesta de ubicación incluye el identificador único del objeto.
202	Aceptado	Se ha iniciado un proceso en segundo plano de larga duración para realizar la solicitud, pero la operación aún no ha finalizado.
400	Solicitud incorrecta	La solicitud introducida no se reconoce o es inapropiada.
403	Prohibido	El acceso está denegado debido a un error de autorización.
404	No encontrado	El recurso al que se hace referencia en la solicitud no existe.
405	Método no permitido	El verbo HTTP en la solicitud no es compatible con el recurso.
409	Conflicto	Se ha producido un intento de crear un objeto, pero el objeto ya existe.
500	Error interno	Se ha producido un error interno general en el servidor.
501	No implementado	Se conoce la URI, pero no es capaz de realizar la solicitud.

Encabezados de respuesta

La respuesta HTTP generada por el servidor Deploy incluye varios encabezados, entre ellos:

- ID de solicitud A cada solicitud de API exitosa se le asigna un identificador de solicitud único.
- location Cuando se crea un objeto, el encabezado de location incluye la URL completa al nuevo objeto, incluido el identificador único del objeto.

Cuerpo de respuesta

El contenido de la respuesta asociada a una solicitud de API varía según el objeto, el tipo de procesamiento y el éxito o fracaso de la solicitud. El cuerpo de la respuesta se muestra en formato JSON.

- Objeto único: se puede devolver un único objeto con un conjunto de campos según la solicitud. Por ejemplo, puedes usar GET para recuperar propiedades seleccionadas de un clúster usando el identificador único.
- Se pueden devolver varios objetos de una colección de recursos. En todos los casos, se utiliza un formato consistente, donde `num_records` indica el número de registros y los registros contienen una matriz de instancias de objetos. Por ejemplo, se pueden recuperar todos los nodos definidos en un clúster específico.
- Objeto Job Si una llamada a la API se procesa de forma asíncrona, se devuelve un objeto Job que sirve de ancla para la tarea en segundo plano. Por ejemplo, la solicitud POST utilizada para implementar un clúster se procesa de forma asíncrona y devuelve un objeto Job.
- Objeto Error Si ocurre un error, siempre se devuelve un objeto Error. Por ejemplo, recibirás un error al intentar crear un clúster con un nombre que ya existe.
- En ciertos casos, no se devuelve ningún dato 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 asíncrono mediante el objeto Job para ONTAP Select

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

Solicitudes asíncronas descritas mediante el objeto Job

Tras realizar una llamada a la API que se ejecuta de forma asíncrona, 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 tras la respuesta HTTP inicial al cliente. La respuesta incluye el objeto Job que ancla la solicitud, incluido su identificador único.

> Deberías consultar la página de documentación online de ONTAP Select Deploy para determinar qué llamadas a la API de REST funcionan de forma asíncrona.

Consultar el objeto Job asociado a una solicitud de API

El objeto Job devuelto en la respuesta HTTP contiene varias propiedades. Puedes consultar la propiedad state para determinar si la solicitud se ha completado correctamente. Un objeto Job puede encontrarse en uno de los siguientes estados:

- En cola
- En ejecución
- Éxito
- Fallo

Hay dos técnicas que puedes usar cuando sondeas un objeto Job para detectar un estado terminal de 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 solo se devuelve cuando ocurre una de las siguientes situaciones:
  - El estado ha cambiado más recientemente que el valor de fecha-hora proporcionado en la solicitud de sondeo
  - El tiempo de espera ha expirado (1 a 120 segundos)

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



Siempre debes usar el sondeo prolongado para reducir la carga de trabajo en la máquina virtual Deploy.

Procedimiento general para emitir una solicitud asíncrona

Puedes usar el siguiente procedimiento a grandes rasgos para completar una llamada asíncrona a la API:

1. Emite la llamada asíncrona a la API.
2. Recibe una respuesta HTTP 202 indicando la aceptación satisfactoria de la solicitud.
3. Extrae el identificador del objeto Job del cuerpo de la respuesta.
4. Dentro de un bucle, realiza lo siguiente en cada ciclo:
   a. Obtén el estado actual del Job con una petición long-poll
   b. Si el trabajo se encuentra en un estado no terminal (en cola, en ejecución), realiza el bucle de nuevo.
5. Detén cuando el Job alcanza un estado terminal (éxito, fracaso).

Accede con un navegador

Descubre cómo acceder a la API de ONTAP Select Deploy desde un navegador

Hay varias cosas que debes tener en cuenta antes de usar la página de documentación en línea de Deploy.

Plan de despliegue

Si tienes intención de realizar llamadas a la API como parte de tareas específicas de despliegue o administración, deberías considerar crear un plan de despliegue. Estos planes pueden ser formales o informales y, en general, contienen tus objetivos y las llamadas a la API que se van a usar. Consulta los procesos de flujo de trabajo usando la API de REST de Deploy para más información.

Ejemplos y definiciones de parámetros JSON

Cada llamada a la API se describe en la página de documentación utilizando un formato coherente. El contenido incluye notas de implementación, parámetros de consulta y códigos de estado HTTP. Además, puedes mostrar detalles sobre el JSON utilizado con las solicitudes y respuestas de la API de la siguiente manera:

- Valor de ejemplo Si seleccionas *Valor de ejemplo* en una llamada a la API, se muestra una estructura

JSON típica para la llamada. Puedes modificar el ejemplo según sea necesario y usarlo como entrada para tu solicitud.

- Modelo Si seleccionas *Modelo*, se muestra una lista completa de los parámetros JSON, con una descripción para cada parámetro.

Precaución al realizar llamadas a la API

Todas las operaciones de la API que realices utilizando la página de documentación de Deploy son operaciones en directo. Debes tener cuidado de no crear, actualizar o eliminar por error la configuración u otros datos.

Accede a la página de documentación de ONTAP Select Deploy

Debes acceder a la página de documentación en línea de ONTAP Select Deploy para visualizar la documentación de la API, así como para realizar una llamada a la API manualmente.

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 selecciona **Intro**:

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

2. Inicia sesión con el nombre de usuario y la contraseña de administrador.

Resultado

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

Comprende y ejecuta una llamada a la API de ONTAP Select Deploy

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

Antes de empezar

Debes iniciar sesión en la página web de documentación en línea de ONTAP Select Deploy. Debes tener el identificador único que se asignó a tu clúster de ONTAP Select cuando se creó el clúster.

Acerca de esta tarea

Puedes recuperar la información de configuración que describe un clúster ONTAP Select usando su identificador único. En este ejemplo, se devuelven todos los campos clasificados como de bajo costo. Sin embargo, como mejor práctica, deberías solicitar solo los campos específicos que necesites.

Pasos

1. En la página principal, desplázate hasta la parte inferior y selecciona **Cluster**.
2. Selecciona **GET /clusters/{cluster_id}** para mostrar los detalles de la llamada a la API utilizada para devolver información sobre un cluster de ONTAP Select.

Procesos de flujo de trabajo

Infórmate sobre cómo usar los flujos de trabajo de la API de ONTAP Select Deploy

Deberías prepararte para revisar y usar los procesos de flujo de trabajo.

Comprende las llamadas a la 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 de REST. En lugar de repetir esos detalles aquí, cada llamada a la API utilizada en los ejemplos de flujos de trabajo incluye solo la información que necesitas para localizar la llamada en la página de documentación. Después de localizar una llamada a la API específica, puedes revisar los detalles completos de la llamada, incluidos 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 a la API dentro de un flujo de trabajo para ayudar a localizar la llamada en la página de documentación:

- Categoría Las llamadas a la API están organizadas en la página de documentación en áreas o categorías relacionadas funcionalmente. Para localizar una llamada API específica, desplázate hasta la parte inferior de la página y selecciona la categoría API aplicable.
- Verbo HTTP El verbo HTTP identifica la acción realizada sobre 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 agrega a la URL principal para formar la URL completa que identifica el recurso.

Construye una URL para acceder directamente a la API de REST

Además de la página de documentación de ONTAP Select, también puedes acceder a la API de REST de Deploy directamente mediante un lenguaje de programación como Python. En este caso, la URL principal es ligeramente diferente a la URL utilizada cuando accedes a la página de documentación en línea. Cuando accedes a la API directamente, debes agregar /api a la cadena de dominio y puerto. Por ejemplo: `http://deploy.mycompany.com/api`

Crear un clúster de evaluación de ONTAP Select de un solo nodo en ESXi

Puedes desplegar un clúster de un solo nodo ONTAP Select en un host VMware ESXi gestionado por vCenter. El clúster se crea con una licencia de evaluación.

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

- El host ESXi no está gestionado por vCenter (host independiente).
- Dentro del clúster se utilizan varios nodos u hosts.
- El clúster está implementado en un entorno de producción con una licencia adquirida.
- Se utiliza el hipervisor KVM en lugar de VMware ESXi.

Paso 1: Registrar la credencial del servidor vCenter

Al realizar una implementación en un host ESXi gestionado por un servidor vCenter, necesitas añadir una credencial antes de registrar el host. Luego, la utilidad de administración Deploy puede usar la credencial para autenticarse en vCenter.

Categoría	Verbo HTTP	Ruta
Desplegar	POST	/seguridad/credenciales

Curl

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

Entrada JSON (step01)

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

Tipo de procesamiento

Asíncrono

Salida

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

Paso 2: Registrar un host de hipervisor

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

Categoría	Verbo HTTP	Ruta
Clúster	POST	/hosts

Curl

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

Entrada JSON (step02)

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

Tipo de procesamiento

Asíncrono

Salida

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

Paso 3: crea un clúster

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

Categoría	Verbo HTTP	Ruta
Clúster	POST	/clusters

Curl

Debes establecer el parámetro de consulta node_count 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 (step03)

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

Tipo de procesamiento

Sincrónico

Salida

- ID del clúster en el encabezado de respuesta de ubicación

Paso 4: configura el clúster

Debes especificar los siguientes atributos como parte de la configuración del clúster.

Categoría	Verbo HTTP	Ruta
Clúster	PATCH	/clusters/{cluster_id}

Curl

Indica 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 (step04)

```
{
  "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

Salida

Ninguno

Paso 5: obtener el nombre de nodo

La utilidad de administración Deploy genera automáticamente los identificadores y los nombres de los nodos al crear un clúster. Antes de poder configurar un nodo, necesitas obtener el ID asignado.

Categoría	Verbo HTTP	Ruta
Clúster	GET	/clusters/{cluster_id}/nodes

Curl

Indica 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

Salida

- Los registros de la matriz describen cada uno un único nodo con el ID y el nombre únicos.

Paso 6: configura los nodos

Proporciona la configuración básica del nodo, que es la primera de las tres llamadas a la API que se utilizan para configurar un nodo.

Categoría	Verbo HTTP	Ruta
Clúster	RUTA	/clusters/{cluster_id}/nodes/{node_id}

Curl

Indica 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 (step06)

Indica el ID del host en el que se ejecutará el nodo ONTAP Select.

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

Tipo de procesamiento

Sincrónico

Salida

Ninguno

Paso 7: Recuperar las redes de nodos

Identifica las redes de datos y de gestión 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	Ruta
Clúster	GET	/clusters/{cluster_id}/nodes/{node_id}/networks

Curl

Indica 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

Salida

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

Paso 8: Configura la red del nodo

Configura las redes de datos y de gestión. La red interna no se utiliza en un clúster de un solo nodo.

Realiza la siguiente llamada a la API dos veces, una vez por cada red.

Categoría	Verbo HTTP	Ruta
Clúster	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

Indica el ID del clúster, el ID del nodo y el ID de 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 (step08)

Indica el nombre de la red.

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

Tipo de procesamiento

Sincrónico

Salida

Ninguno

Paso 9: Configura el grupo de almacenamiento del nodo

El último paso en la configuración de un nodo es adjuntar un pool de almacenamiento. Puedes determinar los pools de almacenamiento disponibles a través del cliente web vSphere o, opcionalmente, a través de la API de REST de Deploy.

Categoría	Verbo HTTP	Ruta
Clúster	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

Indica el ID del clúster, el ID del nodo y el ID de 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 (step09)

La capacidad del pool es de 2 TB.

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

Tipo de procesamiento

Sincrónico

Salida

Ninguno

Paso 10: implementa el clúster

Una vez configurados el clúster y el nodo, puedes desplegar el clúster.

Categoría	Verbo HTTP	Ruta
Clúster	POST	/clusters/{cluster_id}/deploy

Curl

Indica 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 (step10)

Introduce la contraseña de la cuenta de administrador de ONTAP.

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

Tipo de procesamiento

Asíncrono

Salida

- Objeto de trabajo

Información relacionada

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

Acceso con Python

Descubre cómo acceder a la API de ONTAP Select Deploy mediante Python

Debes preparar el entorno antes de ejecutar los scripts de ejemplo de Python.

Antes de ejecutar los scripts de Python, debes asegurarte de que el entorno está configurado correctamente:

- Debe estar instalada la última versión aplicable de Python2. Los códigos de ejemplo han sido probados usando Python2. También deberían ser portables a Python3, pero no se ha probado su compatibilidad.
- Las bibliotecas Requests y urllib3 deben estar instaladas. Puedes usar pip u otra herramienta de gestión de Python según convenga a tu entorno.
- La estación de trabajo cliente donde se ejecutan los scripts debe tener acceso de red a la máquina virtual de ONTAP Select Deploy.

Además, debes tener la siguiente información:

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

Infórmate sobre los scripts de Python para ONTAP Select Deploy

Los scripts de Python de ejemplo te permiten realizar varias tareas diferentes. Deberías entender los scripts antes de usarlos en una instancia de Deploy en producción.

Características de diseño comunes

Los guiones se han diseñado con las siguientes características comunes:

- Ejecuta desde la interfaz de línea de comandos en una máquina cliente. Puedes ejecutar los scripts de

Python desde cualquier máquina cliente configurada correctamente. Consulta *Antes de comenzar* para más información.

- Acepta parámetros de entrada en la CLI Cada script se controla en la CLI mediante parámetros de entrada.
- Lee el archivo de entrada. Cada script lee un archivo de entrada según su propósito. Al crear o eliminar un clúster, tienes que proporcionar un archivo de configuración JSON. Al agregar una licencia de nodo, tienes que proporcionar un archivo de licencia válido.
- Usa un módulo de soporte común. El módulo de soporte común *deploy_requests.py* contiene una sola clase. Se importa y se usa en cada uno de los scripts.

Crear un clúster

Puedes crear un clúster ONTAP Select utilizando el script cluster.py. Según los parámetros de la línea de comandos y el contenido del archivo de entrada JSON, puedes modificar el script para adaptarlo a tu entorno de implementación de la siguiente manera:

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

Los parámetros de entrada de la línea de comandos para el script incluyen:

- Nombre de host o dirección IP del servidor de implementación
- Contraseña para la cuenta de usuario admin
- Nombre del archivo de configuración JSON
- Indicador detallado para la salida del mensaje

Agregar una licencia de nodo

Si decides implementar un clúster de producción, debes agregar una licencia para cada nodo usando el script *add_license.py*. Puedes agregar la licencia antes o después de implementar el clúster.

Los parámetros de entrada de la línea de comandos para el script incluyen:

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

Eliminar un clúster

Puedes eliminar un clúster ONTAP Select existente utilizando el script *delete_cluster.py*.

Los parámetros de entrada de la línea de comandos para el script incluyen:

- Nombre de host o dirección IP del servidor de implementación

- Contraseña para la cuenta de usuario admin
- Nombre del archivo de configuración JSON

Ejemplos de código Python

Script para crear un clúster ONTAP Select

Puedes usar 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 el script para crear un clúster ONTAP Select

Al crear o eliminar un clúster de ONTAP Select mediante los ejemplos de código Python, debes proporcionar un archivo JSON como entrada al script. Puedes copiar y modificar el ejemplo JSON adecuado según tus planes de despliegue.

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 usando 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 añadir una licencia de nodo ONTAP Select

Puedes usar 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

Puedes usar 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 Python de soporte común para ONTAP Select

Todos los scripts Python utilizan una clase Python común en un único 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 redimensionar los nodos del clúster ONTAP Select

Puedes usar el siguiente script para redimensionar 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())
```

Información de copyright

Copyright © 2026 NetApp, Inc. Todos los derechos reservados. Imprimido en EE. UU. No se puede reproducir este documento protegido por copyright ni parte del mismo de ninguna forma ni por ningún medio (gráfico, electrónico o mecánico, incluidas fotocopias, grabaciones o almacenamiento en un sistema de recuperación electrónico) sin la autorización previa y por escrito del propietario del copyright.

El software derivado del material de NetApp con copyright está sujeto a la siguiente licencia y exención de responsabilidad:

ESTE SOFTWARE LO PROPORCIONA NETAPP «TAL CUAL» Y SIN NINGUNA GARANTÍA EXPRESA O IMPLÍCITA, INCLUYENDO, SIN LIMITAR, LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN CONCRETO, CUYA RESPONSABILIDAD QUEDA EXIMIDA POR EL PRESENTE DOCUMENTO. EN NINGÚN CASO NETAPP SERÁ RESPONSABLE DE NINGÚN DAÑO DIRECTO, INDIRECTO, ESPECIAL, EJEMPLAR O RESULTANTE (INCLUYENDO, ENTRE OTROS, LA OBTENCIÓN DE BIENES O SERVICIOS SUSTITUTIVOS, PÉRDIDA DE USO, DE DATOS O DE BENEFICIOS, O INTERRUPCIÓN DE LA ACTIVIDAD EMPRESARIAL) CUALQUIERA SEA EL MODO EN EL QUE SE PRODUJERON Y LA TEORÍA DE RESPONSABILIDAD QUE SE APLIQUE, YA SEA EN CONTRATO, RESPONSABILIDAD OBJETIVA O AGRAVIO (INCLUIDA LA NEGLIGENCIA U OTRO TIPO), QUE SURJAN DE ALGÚN MODO DEL USO DE ESTE SOFTWARE, INCLUSO SI HUBIEREN SIDO ADVERTIDOS DE LA POSIBILIDAD DE TALES DAÑOS.

NetApp se reserva el derecho de modificar cualquiera de los productos aquí descritos en cualquier momento y sin aviso previo. NetApp no asume ningún tipo de responsabilidad que surja del uso de los productos aquí descritos, excepto aquello expresamente acordado por escrito por parte de NetApp. El uso o adquisición de este producto no lleva implícita ninguna licencia con derechos de patente, de marcas comerciales o cualquier otro derecho de propiedad intelectual de NetApp.

Es posible que el producto que se describe en este manual esté protegido por una o más patentes de EE. UU., patentes extranjeras o solicitudes pendientes.

LEYENDA DE DERECHOS LIMITADOS: el uso, la copia o la divulgación por parte del gobierno están sujetos a las restricciones establecidas en el subpárrafo (b)(3) de los derechos de datos técnicos y productos no comerciales de DFARS 252.227-7013 (FEB de 2014) y FAR 52.227-19 (DIC de 2007).

Los datos aquí contenidos pertenecen a un producto comercial o servicio comercial (como se define en FAR 2.101) y son propiedad de NetApp, Inc. Todos los datos técnicos y el software informático de NetApp que se proporcionan en este Acuerdo tienen una naturaleza comercial y se han desarrollado exclusivamente con fondos privados. El Gobierno de EE. UU. tiene una licencia limitada, irrevocable, no exclusiva, no transferible, no sublicenciable y de alcance mundial para utilizar los Datos en relación con el contrato del Gobierno de los Estados Unidos bajo el cual se proporcionaron los Datos. Excepto que aquí se disponga lo contrario, los Datos no se pueden utilizar, desvelar, reproducir, modificar, interpretar o mostrar sin la previa aprobación por escrito de NetApp, Inc. Los derechos de licencia del Gobierno de los Estados Unidos de América y su Departamento de Defensa se limitan a los derechos identificados en la cláusula 252.227-7015(b) de la sección DFARS (FEB de 2014).

Información de la marca comercial

NETAPP, el logotipo de NETAPP y las marcas que constan en http://www.netapp.com/TM son marcas comerciales de NetApp, Inc. El resto de nombres de empresa y de producto pueden ser marcas comerciales de sus respectivos propietarios.