Automatice con REPOSO

ONTAP Select

NetApp
February 03, 2026

Tabla de contenidos

Automatice con REPOSO

Conceptos

Base de servicios web DE REST

La transferencia de estado representacional (REST) es un estilo para crear aplicaciones web distribuidas. Cuando se aplica al diseño de una API de servicios web, establece un conjunto de tecnologías y prácticas recomendadas para exponer recursos basados en servidor y administrar sus estados. Usa los protocolos y estándares más habituales para proporcionar una base flexible para la implementación y gestión de clústeres de ONTAP Select.

Con las restricciones clásicas y a la arquitectura

REST fue formalmente articulado por Roy Fielding en su doctorado "disertación" en UC Irvine en 2000. Define un estilo arquitectónico a través de un conjunto de restricciones, que han mejorado colectivamente las aplicaciones basadas en web y los protocolos subyacentes. Estas restricciones establecen una aplicación de servicios web RESTful basada en una arquitectura de cliente/servidor que utiliza un protocolo de comunicación sin estado.

Recursos y representación estatal

Los recursos son los componentes básicos de un sistema basado en la Web. Al crear una aplicación DE SERVICIOS web DE REST, las tareas de diseño más tempranas incluyen:

- Identificación de los recursos basados en el sistema o en el servidor cada sistema utiliza y mantiene los recursos. Un recurso puede ser un archivo, una transacción comercial, un proceso o una entidad administrativa. Una de las primeras tareas en el diseño de una aplicación basada en servicios web DE REST es identificar los recursos.
- Definición de estados de recursos y operaciones estatales asociadas los recursos siempre se encuentran en uno de un número finito de estados. Los estados, así como las operaciones asociadas utilizadas para afectar los cambios de estado, deben estar claramente definidos.

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

Extremos de URI

Todos los recursos REST deben definirse y ponerse a disposición mediante un esquema de direccionamiento bien definido. Los extremos en los que se encuentran 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 de la red. El Localizador uniforme de recursos (URL) es un tipo de URI que se utiliza con los servicios web para identificar y acceder a los recursos. Los recursos normalmente se exponen en una estructura jerárquica similar a un directorio de archivos.

Mensajes HTTP

El Protocolo de transferencia de hipertexto (HTTP) es el protocolo utilizado por el cliente y 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 administración de estado correspondientes.

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

Formato JSON

Aunque la información se puede estructurar y transferir entre un cliente y un servidor de varias maneras, la opción más popular (y la que se usa con la API REST de deploy) es la notación de objetos JavaScript (JSON). JSON es un estándar del sector para representar estructuras de datos simples en texto sin formato y se utiliza para transferir información de estado que describe los recursos.

Cómo acceder a la API de implementación

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

Implemente la interfaz de usuario nativa de la utilidad

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

Página de documentación en línea de ONTAP Select Deploy

La página de documentación en línea ONTAP Select Deploy proporciona un punto de acceso alternativo cuando se utiliza un explorador. Además de proporcionar una forma de ejecutar directamente llamadas API individuales, la página también incluye una descripción detallada de la API, incluidos los parámetros de entrada y otras opciones para cada llamada. Las llamadas API se organizan en varias categorías o áreas funcionales diferentes.

Programa personalizado

Puede acceder a la API de implementación utilizando cualquiera de los diferentes lenguajes y herramientas de programación. Entre las opciones más populares se incluyen Python, Java y curl. Programa, script o herramienta que usa la API actúa como cliente DE servicios web REST. Utilizar un lenguaje de programación le permite comprender mejor la API y proporciona una oportunidad para automatizar las puestas en marcha de ONTAP Select.

Implemente el control de versiones de API

La API DE REST que se incluye con la implementación de ONTAP Select tiene asignado un número de versión. El número de versión de la API es independiente del número de versión de implementación. Debe conocer la versión de la API que se incluye con su versión de implementación y cómo esto puede afectar al uso que hace de la API.

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

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

Puesta en marcha 2.7.2 y anteriores

ONTAP Select Deploy 2.7.2 y todas las versiones anteriores incluyen la versión 2 de la API DE 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 incluya la versión 2 de la API, debe actualizar cualquier código existente que acceda directamente a la API, así como cualquier script que utilice la interfaz de línea de comandos.

Características operativas básicas

Mientras QUE REST establece un conjunto común de tecnologías y prácticas recomendadas, los detalles de cada API pueden variar en función de las opciones de diseño. Debe tener en cuenta los detalles y las características operativas de la API de implementación de ONTAP Select antes de usar la API.

Host de hipervisor frente a nodo ONTAP Select

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

Identificadores de objeto

A cada instancia u objeto de recurso se le asigna un identificador único cuando se crea. Estos identificadores son globalmente únicos dentro de una instancia específica de la implementación de ONTAP Select. Después de emitir una llamada API que crea una nueva instancia de objeto, el valor de ID asociado se devuelve al llamador en `location` la cabecera de la respuesta HTTP. Puede extraer el identificador y utilizarlo en llamadas posteriores cuando haga referencia a la instancia del recurso.

> El contenido y la estructura interna de los identificadores de objeto pueden cambiar en cualquier momento. Solo se deben usar los identificadores en las llamadas API aplicables según sea necesario cuando se hacen referencia a los objetos asociados.

Identificadores de solicitudes

A cada solicitud API de éxito se le asigna un identificador único. El identificador se devuelve en `request-id` la cabecera de la respuesta HTTP asociada. Puede utilizar un identificador de solicitud para hacer referencia colectivamente a las actividades de una única transacción específica de solicitud y respuesta de API. Por ejemplo, puede recuperar todos los mensajes de eventos de una transacción basándose en el ID de solicitud

Llamadas síncronas y asíncronas

Hay dos formas principales de que un servidor realice una solicitud HTTP recibida desde un cliente:

- Síncrono el servidor realiza la solicitud inmediatamente y responde con un código de estado de 200, 201 o 204.
- Asincrónica el servidor acepta la solicitud y responde con un código de estado de 202. Esto indica que el servidor ha aceptado la solicitud de cliente y ha iniciado una tarea en segundo plano para completar la solicitud. El éxito o el fallo final no están disponibles de forma inmediata y se deben determinar mediante

llamadas API adicionales.

Confirmar que se ha completado un trabajo de ejecución prolongada

Por lo general, cualquier operación que puede tardar mucho tiempo en completarse se procesa de forma asíncrona mediante una tarea en segundo plano en el servidor. Con la API Deploy REST, cada tarea en segundo plano está anclada por 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 crear una tarea en segundo plano.

Puede consultar el objeto Job directamente para determinar el éxito o el error de la llamada API asociada. Consulte *procesamiento asíncrono mediante el objeto Job* para obtener información adicional.

Además de utilizar el objeto Job, existen otras formas de determinar el éxito o el fallo de una solicitud, entre las que se incluyen:

- Mensajes de eventos puede recuperar todos los mensajes de eventos asociados con una llamada API específica utilizando el ID de solicitud devuelto con la respuesta original. Los mensajes de eventos normalmente contienen una indicación de éxito o fallo, y también pueden ser útiles al depurar una condición de error.
- Estado o estado de recurso varios de los recursos mantienen un valor de estado o estado al que puede consultar para determinar indirectamente el éxito o el error 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 a través de TLS. No se admite el uso del protocolo HTTP a través de un canal no cifrado. Se admite la versión 1.2 de TLS.
- Autenticación HTTP la autenticación básica se utiliza para cada transacción de API. A cada solicitud se agrega un encabezado HTTP, que incluye el nombre de usuario y la contraseña en una cadena base64.

Transacción de API de solicitud y respuesta

Cada llamada de API de implementación se realiza como una solicitud HTTP a la máquina virtual de implementación, que genera una respuesta asociada al cliente. Este par de solicitud/respuesta se considera una transacción de API. Antes de utilizar la API de implementación, debería estar familiarizado con las variables de entrada disponibles para controlar una solicitud y el contenido del resultado de la respuesta.

Variables de entrada que controlan una solicitud API

Puede controlar cómo se procesa una llamada API mediante parámetros definidos en la solicitud HTTP.

Solicitar encabezados

Debe incluir varios encabezados en la solicitud HTTP, incluidos:

- Tipo de contenido Si el cuerpo de la solicitud incluye JSON, este encabezado debe establecerse en Application/json.
- Acepte Si el cuerpo de respuesta incluirá JSON, este encabezado debe establecerse en Application/json.

- Autorización la autenticación básica se debe establecer con el nombre de usuario y la contraseña codificados en una cadena base64.

Solicitar el cuerpo

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

- Objeto JSON con variables de entrada (como el nombre de un clúster nuevo)
- Vacío

Filtrar objetos

Al emitir una llamada API que utilice GET, puede limitar o filtrar los objetos devueltos en función de cualquier atributo. Por ejemplo, puede especificar un valor exacto para que coincida:

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

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

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

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

Selección de campos de objeto

De forma predeterminada, al emitir una llamada API mediante GET, sólo se devuelven los atributos que identifican de forma exclusiva 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 de objeto adicionales mediante el parámetro de consulta Campos de las siguientes formas:

- Los campos de bajo coste especifican `fields=*` recuperar los campos de objeto que se mantienen en la memoria del servidor local o que requieren poco procesamiento para acceder.
- Los campos caros especifican `fields=**` para recuperar todos los campos de objeto, incluidos los que requieren un procesamiento de servidor adicional para acceder.
- Selección de campo personalizado: Utilice `fields=FIELDNAME` para especificar el campo exacto que desea. Al solicitar varios campos, los valores deben separarse con comas sin espacios.

Como práctica recomendada, siempre debe identificar los campos específicos que desea. Sólo debe recuperar el conjunto de campos baratos o caros cuando sea necesario. NetApp determina la clasificación económica y cara basándose en el análisis del rendimiento interno. 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 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, puede ordenar el campo de tipo en orden descendente seguido de id en orden ascendente:

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

Cuando se incluyan varios parámetros, los campos deben separarse con una coma.

Paginación

Al emitir una llamada API mediante GET para acceder a una colección de objetos del mismo tipo, todos los objetos coincidentes se devuelven de forma predeterminada. Si es necesario, 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 restringir el conjunto de resultados. Por ejemplo, lo siguiente devuelve hasta 10 eventos del sistema generados después del tiempo especificado:

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

Puede emitir varias solicitudes para páginas a través de los eventos (o cualquier tipo de objeto). Cada llamada API posterior debe utilizar un nuevo valor de tiempo basado en el último evento del último conjunto de resultados.

Interpretar una respuesta API

Cada solicitud de API genera una respuesta al cliente. Puede examinar la respuesta para determinar si ha tenido éxito y recuperar datos adicionales según sea necesario.

Código de estado HTTP

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

Codificación	Significado	Descripción
200	DE ACUERDO	Indica que las llamadas que no crean un objeto nuevo se han realizado correctamente.
201	Creado	Se ha creado correctamente un objeto; el encabezado de respuesta de ubicación incluye el identificador único del objeto.
202	Aceptado	Se inició un trabajo en segundo plano de ejecución prolongada para realizar la solicitud, pero la operación aún no se ha completado.
400	Solicitud incorrecta	La entrada de la solicitud no se reconoce o no es apropiada.

Codificación	Significado	Descripción
403	Prohibido	Se deniega el acceso 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 de la solicitud no es compatible con el recurso.
409	Conflicto	Error al intentar crear un objeto porque el objeto ya existe.
500	Error interno	Se ha producido un error interno general en el servidor.
501	No implementada	El URI es conocido 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, entre los que se incluyen:

- ID de solicitud a cada solicitud API correcta se le asigna un identificador de solicitud único.
- Ubicación cuando se crea un objeto, el encabezado de ubicación incluye la dirección URL completa del nuevo objeto, incluido el identificador de objeto único.

Cuerpo de respuesta

El contenido de la respuesta asociada a una solicitud API varía en función del objeto, el tipo de procesamiento y el éxito o el fallo de la solicitud. El cuerpo de la respuesta se representa en JSON.

- Objeto único un solo objeto se puede devolver con un conjunto de campos basados en la solicitud. Por ejemplo, se puede usar GET para recuperar las propiedades seleccionadas de un clúster mediante el identificador único.
- Se pueden devolver varios objetos de una colección de recursos. En todos los casos, se utiliza un formato consistente, con `num_records` la indicación del número de registros y registros que contienen una matriz de las instancias de objeto. Por ejemplo, puede recuperar todos los nodos definidos en un clúster específico.
- Objeto de trabajo Si una llamada API se procesa de forma asíncrona, se devuelve un objeto Job que ancla 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 se produce un error, siempre se devuelve un objeto error. Por ejemplo, recibirá un error al intentar crear un clúster con un nombre que ya existe.
- Vacío en determinados casos, no se devuelve ningún dato y el cuerpo de respuesta está vacío. Por ejemplo, el cuerpo de respuesta está vacío después de utilizar DELETE para eliminar un host existente.

Procesamiento asíncrono mediante el objeto de trabajo

Algunas de las llamadas de implementación de API, especialmente las que crean o modifican un recurso, pueden tardar más tiempo en completarse que otras llamadas. La implementación de ONTAP Select procesa estas solicitudes de ejecución prolongada de forma asíncrona.

Solicitudes asincrónicas descritas mediante el objeto Job

Después de realizar una llamada API que se ejecuta de forma asíncrona, el código de respuesta HTTP 202

indica que la solicitud se ha validado y aceptado correctamente, pero que 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 anclando la solicitud, incluyendo su identificador único.

Consulte la página de documentación en línea de implementación de ONTAP Select para determinar qué llamadas API funcionan de forma asíncrona.

Consulte el objeto Job asociado a una solicitud API

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

- En cola
- Ejecutando
- Correcto
- Fallo

Existen dos técnicas que se pueden utilizar al sondear un objeto Job para detectar un estado de terminal para la tarea, ya sea con éxito o con un error:

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

Sondeo estándar y sondeo largo Utilice la misma llamada API para consultar un objeto Job. Sin embargo, una solicitud de sondeo larga incluye dos parámetros de consulta: `poll_timeout` Y `last_modified`.

Siempre debe utilizar los sondeos largos 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 asíncrona:

1. Emita la llamada de API asíncrona.
2. Reciba una respuesta HTTP 202 que indique la aceptación correcta de la solicitud.
3. Extraiga el identificador del objeto Job del cuerpo de respuesta.
4. Dentro de un bucle, realice lo siguiente en cada ciclo:
   a. Obtener el estado actual del trabajo con una solicitud de sondeo largo
   b. Si el trabajo se encuentra en un estado que no es terminal (en cola, en ejecución), vuelva a realizar el bucle.
5. Deténgase cuando el trabajo alcance un estado terminal (correcto, fallo).

Acceso con un explorador

Antes de acceder a la API con un explorador

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

Plan de implementación

Si piensa emitir llamadas API como parte de la realización de tareas administrativas o de implementación específicas, debería considerar la creación de un plan de implementación. Estos planes pueden ser formales o informales, y, por lo general, contienen sus objetivos y las llamadas a las API que se deben utilizar. Consulte procesos de flujo de trabajo mediante la API DE REST de puesta en marcha para obtener más información.

Ejemplos de JSON y definiciones de parámetros

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

- Ejemplo de valor Si hace clic en *ejemplo valor* en una llamada API, se muestra una estructura JSON típica para la llamada. Puede modificar el ejemplo según sea necesario y utilizarlo como entrada para su solicitud.
- Si hace clic en *Model*, se muestra una lista completa de los parámetros JSON, con una descripción de cada parámetro.

Precaución al emitir llamadas API

Todas las operaciones de API que se realizan mediante la página de documentación de implementación son operaciones en directo. Debe tener cuidado de no crear, actualizar o eliminar por error la configuración u otros datos.

Acceda a la página de documentación de despliegue

Para ver la documentación en línea de la implementación de ONTAP Select, debe acceder a la documentación de API, y también para emitir manualmente una llamada API.

Antes de empezar

Debe tener lo siguiente:

- La dirección IP o el nombre de dominio de la máquina virtual de implementación de ONTAP Select
- Nombre de usuario y contraseña del administrador

Pasos

1. Escriba la dirección URL en su navegador y pulse **Intro**:

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

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

Resultado

Se muestra la página web de la documentación de despliegue con las llamadas organizadas por categoría en la parte inferior de la página.

Entender y ejecutar una llamada API

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

Antes de empezar

Debe haber iniciado sesión en la página web de documentación en línea de implementación de ONTAP Select. Debe tener asignado el identificador único al clúster ONTAP Select cuando se creó el clúster.

Acerca de esta tarea

Puede recuperar la información de configuración que describe un clúster de ONTAP Select con su identificador único. En este ejemplo, se devuelven todos los campos clasificados como baratos. Sin embargo, como práctica recomendada, solo se deben solicitar los campos específicos que se necesitan.

Pasos

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

Procesos de flujo de trabajo

Antes de usar los flujos de trabajo de la API

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 DE REST. En lugar de repetir estos detalles aquí, cada llamada de API utilizada en los ejemplos de flujo de trabajo incluye solo la información necesaria para localizar la llamada en la página de documentación. Después de localizar una llamada API específica, puede revisar los detalles completos de la llamada, incluidos los parámetros de entrada, formatos de salida, códigos de estado HTTP y tipo de procesamiento de solicitudes.

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

- Categoría las llamadas API se organizan en la página de documentación en categorías o áreas relacionadas con la funcionalidad. Para ubicar una llamada API específica, desplácese hasta la parte inferior de la página y haga clic en la categoría API correspondiente.
- Verbo HTTP el verbo HTTP identifica la acción realizada en un recurso. Cada llamada API se ejecuta a través de un único verbo HTTP.
- Ruta de acceso la ruta de acceso determina el recurso específico al que se aplica la acción como parte de la realización de una llamada. La cadena de ruta de acceso se anexa a la URL de núcleo para formar la

URL completa que identifica el recurso.

Construir una URL para acceder directamente a la API de REST

Además de la página de documentación de ONTAP Select, también puede acceder a la API DE REST de puesta en marcha directamente mediante un lenguaje de programación como Python. En este caso, la URL principal es ligeramente diferente a la URL utilizada al acceder a la página de documentación en línea. Al acceder a la API directamente, debe anexar /api al dominio y la cadena de puerto. Por ejemplo: `http://deploy.mycompany.com/api`

Flujo de trabajo 1: Cree un clúster de evaluación de un solo nodo en ESXi

Se puede poner en marcha un clúster de ONTAP Select de un solo nodo en un host VMware ESXi gestionado por vCenter. El clúster se crea con una licencia de evaluación.

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

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

1. Registre la credencial de vCenter Server

Cuando se ponga en marcha en un host ESXi gestionado por una instancia de vCenter Server, debe añadir una credencial antes de registrar el host. La utilidad de administración de implementación puede usar la credencial para autenticar en vCenter.

Categoría	Verbo HTTP	Ruta
Puesta en marcha	PUBLICAR	/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

Asíncrona

Salida

- ID de credencial en la cabecera de respuesta de ubicación
- Objeto de trabajo

2. Registre un host de hipervisor

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

Categoría	Verbo HTTP	Ruta
Clúster	PUBLICAR	/hosts

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

Asíncrona

Salida

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

3. Cree un clúster

Cuando se crea un clúster ONTAP Select, se registra la configuración básica de clúster y los nombres de los nodos se generan automáticamente mediante la implementación.

Categoría	Verbo HTTP	Ruta
Clúster	PUBLICAR	/cluster

Rizo

El parámetro de consulta node_count se debe establecer como 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

Síncrona

Salida

- ID de clúster en el encabezado de respuesta de ubicación

4. Configure el clúster

Debe proporcionar varios atributos como parte de la configuración del clúster.

Categoría	Verbo HTTP	Ruta
Clúster	PARCHE	/cluster/{cluster_id}

Rizo

Debe proporcionar el ID de 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

Síncrona

Salida

Ninguno

5. Recupere el nombre del nodo

La utilidad de administración Deploy genera automáticamente los identificadores de nodo y los nombres cuando se crea un clúster. Para poder configurar un nodo, debe recuperar el ID asignado.

Categoría	Verbo HTTP	Ruta
Clúster	OBTENGA	/cluster/{cluster_id}/nodos

Rizo

Debe proporcionar el ID de clúster.

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

Tipo de procesamiento

Síncrona

Salida

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

6. Configure los nodos

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

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

Rizo

Debe proporcionar el ID de clúster y el ID de 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 de host donde se ejecutará el nodo de ONTAP Select.

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

Tipo de procesamiento

Síncrona

Salida

Ninguno

7. Recupere las redes del nodo

En el clúster de un único nodo, debe identificar las redes de datos y gestión que utiliza el nodo. La red interna no se usa con un clúster de un solo nodo.

Categoría	Verbo HTTP	Ruta
Clúster	OBTENGA	/cluster/{cluster_id}/nodes/{node_id}/redes

Rizo

Debe proporcionar el ID de clúster y el ID de 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

Síncrona

Salida

- Matriz de dos registros que describen una sola red para el nodo, incluyendo el identificador único y el propósito

8. Configure la conexión a redes del nodo

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

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

Categoría	Verbo HTTP	Ruta
Clúster	PARCHE	/cluster/{cluster_id}/nodes/{node_id}/networks/{network_id}

Rizo

Debe proporcionar el ID de clúster, el ID de 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 (paso 08)

Debe proporcionar el nombre de la red.

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

Tipo de procesamiento

Síncrona

Salida

Ninguno

9. Configure el pool de almacenamiento del nodo

El paso final de configurar un nodo es conectar un pool de almacenamiento. Se pueden determinar los pools de almacenamiento disponibles a través del cliente web de vSphere, o bien, de manera opcional, mediante la API DE REST Deploy.

Categoría	Verbo HTTP	Ruta
Clúster	PARCHE	/cluster/{cluster_id}/nodes/{node_id}/networks/{network_id}

Rizo

Debe proporcionar el ID de clúster, el ID de 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 (paso 09)

La capacidad del pool es 2 TB.

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

Tipo de procesamiento

Síncrona

Salida

Ninguno

10. Ponga en marcha el clúster

Después de configurar el clúster y el nodo, puede implementar el clúster.

Categoría	Verbo HTTP	Ruta
Clúster	PUBLICAR	/cluster/{cluster_id}/deploy

Rizo

Debe proporcionar el ID de 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 de la cuenta de administrador de ONTAP.

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

Tipo de procesamiento

Asíncrona

Salida

- Objeto de trabajo

Acceso con Python

Antes de acceder a la API mediante Python

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

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

- Debe instalarse la última versión aplicable de python2. Los códigos de las muestras se han probado utilizando python2. También deben ser portátiles a Python3, pero no han sido probados para la compatibilidad.
- Deben instalarse las solicitudes y las bibliotecas urllib3. Puede utilizar pip u otra herramienta de gestión 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, debe tener la siguiente información:

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

Entiende los scripts de Python

Los scripts Python de ejemplo le permiten realizar varias tareas diferentes. Debe comprender los scripts antes de utilizarlos en una instancia de despliegue en directo.

Características de diseño comunes

Los scripts se han diseñado 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 configurado correctamente. Consulte *antes de comenzar* para obtener más información.
- Aceptar los parámetros de entrada de la 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. Cuando crea o elimina un clúster, debe proporcionar un archivo de configuración JSON. Al añadir una licencia de nodo, debe proporcionar un archivo de licencia válido.
- Utilice un módulo de soporte común el módulo de soporte común *deploy_Requests.py* contiene una sola clase. Cada uno de los scripts lo importa y lo utiliza.

Cree un clúster

Es posible 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 modificar el script en el entorno de implementación de la manera siguiente:

- Hipervisor puede ponerse en marcha en ESXI o KVM (según la versión de puesta en marcha). Cuando se pone en marcha en ESXi, el hipervisor puede gestionarse con vCenter o puede ser un host independiente.
- Tamaño del clúster puede poner en marcha un clúster de un solo nodo o de varios nodos.

- Licencia de evaluación o producción puede implementar un clúster con una evaluación o adquirir una licencia para producción.

Los parámetros de entrada de la CLI para el script incluyen:

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

Añada una licencia de nodo

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

Los parámetros de entrada de la CLI para el script incluyen:

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

Elimine un clúster

Es posible eliminar un clúster ONTAP Select existente con el script *delete_cluster.py*.

Los parámetros de entrada de la CLI para el script incluyen:

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

Muestras de código Python

Script para crear un clúster

Puede utilizar el siguiente script para crear un clúster basado en los parámetros definidos en el script y en 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 a fin de crear un clúster

Cuando crea o elimina un clúster de ONTAP Select con ejemplos de código Python, debe proporcionar un archivo JSON como entrada del script. Puede copiar y modificar la muestra JSON adecuada en función de 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 añadir una licencia de nodo

Se puede usar el siguiente script para añadir una licencia de 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

Es posible utilizar el siguiente script de la 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 soporte común

Todos los scripts de 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 cambiar el tamaño de los nodos del clúster

Puede usar el siguiente script para cambiar el tamaño de los nodos de 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())
```