# Automatize com REST

## ONTAP Select

NetApp
January 29, 2026

# Índice

# Automatize com REST

## Conceitos

### Base de serviços web REST para implantação e gerenciamento de clusters ONTAP Select

A Transferência de Estado Representacional (REST) é um estilo para a criação de aplicações web distribuídas. Quando aplicada ao design de uma API de serviços web, ela estabelece um conjunto de tecnologias e práticas recomendadas para expor recursos baseados em servidor e gerenciar seus estados. Ela utiliza protocolos e padrões tradicionais para fornecer uma base flexível para a implantação e o gerenciamento de clusters ONTAP Select .

#### Arquitetura e restrições clássicas

O REST foi formalmente articulado por Roy Fielding em seu doutorado "dissertação" na UC Irvine em 2000. Ela define um estilo arquitetônico por meio de um conjunto de restrições que, coletivamente, aprimoram aplicações web e os protocolos subjacentes. As restrições estabelecem uma aplicação de serviços web RESTful baseada em uma arquitetura cliente/servidor, utilizando um protocolo de comunicação sem estado.

#### Recursos e representação estatal

Os recursos são os componentes básicos de um sistema web. Ao criar uma aplicação de serviços web REST, as tarefas iniciais de design incluem:

- Identificação de recursos baseados em sistema ou servidor. Todo sistema utiliza e mantém recursos. Um recurso pode ser um arquivo, uma transação comercial, um processo ou uma entidade administrativa. Uma das primeiras tarefas ao projetar uma aplicação baseada em serviços web REST é identificar os recursos.
- Definição de estados de recursos e operações de estado associadas. Os recursos estão sempre em um de um número finito de estados. Os estados, bem como as operações associadas usadas para afetar as mudanças de estado, devem ser claramente definidos.

As mensagens são trocadas entre o cliente e o servidor para acessar e alterar o estado dos recursos de acordo com o modelo genérico CRUD (Criar, Ler, Atualizar e Excluir).

#### Pontos finais de URI

Cada recurso REST deve ser definido e disponibilizado usando um esquema de endereçamento bem definido. Os endpoints onde os recursos estão localizados e identificados usam um Identificador Uniforme de Recursos (URI). O URI fornece uma estrutura geral para a criação de um nome exclusivo para cada recurso na rede. O Localizador Uniforme de Recursos (URL) é um tipo de URI usado com serviços web para identificar e acessar recursos. Os recursos são normalmente expostos em uma estrutura hierárquica semelhante a um diretório de arquivos.

#### Mensagens HTTP

O Protocolo de Transferência de Hipertexto (HTTP) é o protocolo usado pelo cliente e servidor de serviços web para trocar mensagens de solicitação e resposta sobre os recursos. Como parte do projeto de um aplicativo de serviços web, verbos HTTP (como GET e POST) são mapeados para os recursos e as ações de

gerenciamento de estado correspondentes.

O HTTP não possui estado. Portanto, para associar um conjunto de solicitações e respostas relacionadas em uma única transação, informações adicionais devem ser incluídas nos cabeçalhos HTTP transmitidos com os fluxos de dados de solicitação/resposta.

#### Formatação JSON

Embora as informações possam ser estruturadas e transferidas entre um cliente e um servidor de diversas maneiras, a opção mais popular (e a usada com a API REST do Deploy) é a JavaScript Object Notation (JSON). JSON é um padrão do setor para representar estruturas de dados simples em texto simples e é usado para transferir informações de estado que descrevem os recursos.

## Como acessar a API ONTAP Select Deploy

Devido à flexibilidade inerente dos serviços web REST, a API ONTAP Select Deploy pode ser acessada de diversas maneiras diferentes.

#### Implantar interface de usuário nativa do utilitário

A principal maneira de acessar a API é por meio da interface de usuário web do ONTAP Select Deploy. O navegador faz chamadas para a API e reformata os dados de acordo com o design da interface de usuário. Você também acessa a API por meio da interface de linha de comando do utilitário Deploy.

#### Página de documentação on-line do ONTAP Select Deploy

A página de documentação online do ONTAP Select Deploy oferece um ponto de acesso alternativo ao usar um navegador. Além de fornecer uma maneira de executar chamadas de API individuais diretamente, a página também inclui uma descrição detalhada da API, incluindo parâmetros de entrada e outras opções para cada chamada. As chamadas de API são organizadas em diversas áreas ou categorias funcionais.

#### Programa personalizado

Você pode acessar a API de Implantação usando qualquer uma das diversas linguagens de programação e ferramentas. As opções mais populares incluem Python, Java e cURL. Um programa, script ou ferramenta que utiliza a API atua como um cliente de serviços web REST. O uso de uma linguagem de programação permite que você entenda melhor a API e oferece a oportunidade de automatizar as implantações do ONTAP Select .

## Versão da API de implantação do ONTAP Select

A API REST incluída no ONTAP Select Deploy recebe um número de versão. O número da versão da API é independente do número da versão do Deploy. Você deve estar ciente da versão da API incluída na sua versão do Deploy e de como isso pode afetar seu uso da API.

A versão atual do utilitário de administração Deploy inclui a versão 3 da API REST. Versões anteriores do utilitário Deploy incluem as seguintes versões da API:

**Implantar 2.8 e posterior**

ONTAP Select Deploy 2.8 e todas as versões posteriores incluem a versão 3 da API REST.

#### Implantar 2.7.2 e versões anteriores

ONTAP Select Deploy 2.7.2 e todas as versões anteriores incluem a versão 2 da API REST.

> As versões 2 e 3 da API REST não são compatíveis. Se você atualizar para o Deploy 2.8 ou posterior a partir de uma versão anterior que inclua a versão 2 da API, será necessário atualizar todo o código existente que acessa diretamente a API, bem como todos os scripts que usam a interface de linha de comando.

## Características operacionais básicas da API ONTAP Select Deploy

Embora o REST estabeleça um conjunto comum de tecnologias e práticas recomendadas, os detalhes de cada API podem variar de acordo com as escolhas de design. Você deve estar ciente dos detalhes e das características operacionais da API ONTAP Select Deploy antes de usá-la.

#### Host do hipervisor versus ONTAP Select

Um *host hipervisor* é a plataforma de hardware central que hospeda uma máquina virtual ONTAP Select . Quando uma máquina virtual ONTAP Select é implantada e está ativa em um host hipervisor, a máquina virtual é considerada um *nó ONTAP Select*. Com a versão 3 da API REST de Implantação, os objetos host e nó são separados e distintos. Isso permite um relacionamento um-para-muitos, em que um ou mais nós ONTAP Select podem ser executados no mesmo host hipervisor.

#### Identificadores de objetos

Cada instância de recurso ou objeto recebe um identificador exclusivo quando é criado. Esses identificadores são globalmente exclusivos dentro de uma instância específica do ONTAP Select Deploy. Após emitir uma chamada de API que cria uma nova instância de objeto, o valor do ID associado é retornado ao chamador no `location` Cabeçalho da resposta HTTP. Você pode extrair o identificador e usá-lo em chamadas subsequentes ao se referir à instância do recurso.

> O conteúdo e a estrutura interna dos identificadores de objeto podem mudar a qualquer momento. Você deve usar os identificadores somente nas chamadas de API aplicáveis, conforme necessário, ao se referir aos objetos associados.

#### Identificadores de solicitação

Cada solicitação de API bem-sucedida recebe um identificador exclusivo. O identificador é retornado no `request-id` Cabeçalho da resposta HTTP associada. Você pode usar um identificador de solicitação para se referir coletivamente às atividades de uma única transação de solicitação-resposta de API específica. Por exemplo, você pode recuperar todas as mensagens de eventos de uma transação com base no ID da solicitação.

#### Chamadas síncronas e assíncronas

Há duas maneiras principais pelas quais um servidor executa uma solicitação HTTP recebida de um cliente:

- Síncrono O servidor executa a solicitação imediatamente e responde com um código de status 200, 201 ou 204.
- Assíncrono: O servidor aceita a solicitação e responde com o código de status 202. Isso indica que o servidor aceitou a solicitação do cliente e iniciou uma tarefa em segundo plano para concluí-la. O sucesso

ou a falha final não são imediatamente conhecidos e devem ser determinados por meio de chamadas de API adicionais.

#### Confirmar a conclusão de um trabalho de longa duração

Geralmente, qualquer operação que possa levar muito tempo para ser concluída é processada de forma assíncrona usando uma tarefa em segundo plano no servidor. Com a API REST de Implantação, cada tarefa em segundo plano é ancorada por um objeto Job, que rastreia a tarefa e fornece informações, como o estado atual. Um objeto Job, incluindo seu identificador exclusivo, é retornado na resposta HTTP após a criação de uma tarefa em segundo plano.

Você pode consultar o objeto Job diretamente para determinar o sucesso ou a falha da chamada de API associada. Consulte *processamento assíncrono usando o objeto Job* para obter mais informações.

Além de usar o objeto Job, há outras maneiras de determinar o sucesso ou a falha de uma solicitação, incluindo:

- Mensagens de evento: Você pode recuperar todas as mensagens de evento associadas a uma chamada de API específica usando o ID da solicitação retornado com a resposta original. As mensagens de evento geralmente contêm uma indicação de sucesso ou falha e também podem ser úteis na depuração de uma condição de erro.
- Estado ou status do recurso Vários recursos mantêm um valor de estado ou status que você pode consultar para determinar indiretamente o sucesso ou a falha de uma solicitação.

#### Segurança

A API de implantação usa as seguintes tecnologias de segurança:

- Segurança da Camada de Transporte: Todo o tráfego enviado pela rede entre o servidor de implantação e o cliente é criptografado por TLS. O uso do protocolo HTTP em um canal não criptografado não é suportado. A versão 1.2 do TLS é suportada.
- Autenticação HTTP: A autenticação básica é usada para todas as transações da API. Um cabeçalho HTTP, que inclui o nome de usuário e a senha em uma string base64, é adicionado a cada solicitação.

## Transação de API de solicitação e resposta para ONTAP Select

Cada chamada à API de implantação é realizada como uma solicitação HTTP para a máquina virtual de implantação, que gera uma resposta associada ao cliente. Esse par solicitação/resposta é considerado uma transação de API. Antes de usar a API de implantação, você deve estar familiarizado com as variáveis de entrada disponíveis para controlar uma solicitação e o conteúdo da saída da resposta.

#### Variáveis de entrada que controlam uma solicitação de API

Você pode controlar como uma chamada de API é processada por meio de parâmetros definidos na solicitação HTTP.

##### Cabeçalhos de solicitação

Você deve incluir vários cabeçalhos na solicitação HTTP, incluindo:

- content-type Se o corpo da solicitação incluir JSON, este cabeçalho deverá ser definido como application/json.

- aceitar Se o corpo da resposta incluir JSON, este cabeçalho deverá ser definido como application/json.
- autorização A autenticação básica deve ser definida com o nome de usuário e a senha codificados em uma string base64.

#### Corpo da solicitação

O conteúdo do corpo da solicitação varia dependendo da chamada específica. O corpo da solicitação HTTP consiste em um dos seguintes:

- Objeto JSON com variáveis de entrada (como o nome de um novo cluster)
- Vazio

#### Filtrar objetos

Ao emitir uma chamada de API que usa GET, você pode limitar ou filtrar os objetos retornados com base em qualquer atributo. Por exemplo, você pode especificar um valor exato para corresponder a:

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

Além da correspondência exata, existem outros operadores disponíveis para retornar um conjunto de objetos em um intervalo de valores. O ONTAP Select suporta os operadores de filtragem mostrados abaixo.

| Operador | Descrição |
|---|---|
| = | Igual a |
| < | Menor que |
| > | Maior que |
| ⇐ | Menor ou igual a |
| >= | Maior ou igual a |
| \| | Ou |
| ! | Não é igual a |
| * | Curinga ganancioso |

Você também pode retornar um conjunto de objetos com base em se um campo específico está definido ou não, usando a palavra-chave null ou sua negação (!null) como parte da consulta.

#### Selecionando campos de objeto

Por padrão, emitir uma chamada de API usando GET retorna apenas os atributos que identificam exclusivamente o(s) objeto(s). Esse conjunto mínimo de campos atua como uma chave para cada objeto e varia de acordo com o tipo de objeto. Você pode selecionar propriedades adicionais do objeto usando o parâmetro de consulta fields das seguintes maneiras:

- Campos baratos Especificar `fields=*` para recuperar os campos de objeto que são mantidos na memória do servidor local ou que exigem pouco processamento para acesso.
- Campos caros Especificar `fields=**` para recuperar todos os campos do objeto, incluindo aqueles que exigem processamento adicional do servidor para acesso.
- Seleção de campo personalizado Usar `fields=FIELDNAME` para especificar o campo exato desejado. Ao solicitar vários campos, os valores devem ser separados por vírgulas, sem espaços.

Como prática recomendada, você deve sempre identificar os campos específicos que deseja. Você só deve recuperar o conjunto de campos de baixo custo ou alto custo quando necessário. A classificação de baixo custo e alto custo é determinada pela NetApp com base em análises internas de desempenho. A classificação de um determinado campo pode mudar a qualquer momento.

#### Classificar objetos no conjunto de saída

Os registros em uma coleção de recursos são retornados na ordem padrão definida pelo objeto. Você pode alterar a ordem usando o parâmetro de consulta order_by com o nome do campo e a direção de classificação da seguinte maneira:

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

Por exemplo, você pode classificar o campo tipo em ordem decrescente seguido pelo id em ordem crescente:

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

Ao incluir vários parâmetros, você deve separar os campos com uma vírgula.

#### Paginação

Ao emitir uma chamada de API usando GET para acessar uma coleção de objetos do mesmo tipo, todos os objetos correspondentes são retornados por padrão. Se necessário, você pode limitar o número de registros retornados usando o parâmetro de consulta max_records na solicitação. Por exemplo:

```
max_records=20
```

Se necessário, você pode combinar este parâmetro com outros parâmetros de consulta para restringir o conjunto de resultados. Por exemplo, o seguinte retorna até 10 eventos do sistema gerados após o tempo especificado:

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

Você pode emitir várias solicitações para navegar pelos eventos (ou qualquer tipo de objeto). Cada chamada de API subsequente deve usar um novo valor de tempo com base no evento mais recente no último conjunto de resultados.

### Interpretar uma resposta de API

Cada solicitação de API gera uma resposta ao cliente. Você pode examinar a resposta para determinar se foi bem-sucedida e recuperar dados adicionais, se necessário.

#### Código de status HTTP

Os códigos de status HTTP usados pela API REST de implantação são descritos abaixo.

| Código | Significado | Descrição |
|---|---|---|
| 200 | OK | Indica sucesso para chamadas que não criam um novo objeto. |
| 201 | Criado | Um objeto foi criado com sucesso; o cabeçalho de resposta de localização inclui o identificador exclusivo do objeto. |
| 202 | Aceito | Um trabalho em segundo plano de longa execução foi iniciado para executar a solicitação, mas a operação ainda não foi concluída. |
| 400 | Pedido ruim | A entrada solicitada não é reconhecida ou é inadequada. |

| Código | Significado | Descrição |
|---|---|---|
| 403 | Proibido | Acesso negado devido a um erro de autorização. |
| 404 | Não encontrado | O recurso mencionado na solicitação não existe. |
| 405 | Método não permitido | O verbo HTTP na solicitação não é suportado para o recurso. |
| 409 | Conflito | Uma tentativa de criar um objeto falhou porque o objeto já existe. |
| 500 | Erro interno | Ocorreu um erro interno geral no servidor. |
| 501 | Não implementado | O URI é conhecido, mas não é capaz de executar a solicitação. |

#### Cabeçalhos de resposta

Vários cabeçalhos são incluídos na resposta HTTP gerada pelo servidor de implantação, incluindo:

- request-id Cada solicitação de API bem-sucedida recebe um identificador de solicitação exclusivo.
- localização Quando um objeto é criado, o cabeçalho de localização inclui o URL completo para o novo objeto, incluindo o identificador exclusivo do objeto.

#### Corpo de resposta

O conteúdo da resposta associada a uma solicitação de API varia de acordo com o objeto, o tipo de processamento e o sucesso ou falha da solicitação. O corpo da resposta é renderizado em JSON.

- Objeto único: Um único objeto pode ser retornado com um conjunto de campos com base na solicitação. Por exemplo, você pode usar GET para recuperar propriedades selecionadas de um cluster usando o identificador exclusivo.
- Vários objetos Vários objetos de uma coleção de recursos podem ser retornados. Em todos os casos, há um formato consistente usado, com `num_records` indicando o número de registros e registros contendo uma matriz de instâncias do objeto. Por exemplo, você pode recuperar todos os nós definidos em um cluster específico.
- Objeto Job: Se uma chamada de API for processada de forma assíncrona, um objeto Job será retornado, ancorando a tarefa em segundo plano. Por exemplo, a solicitação POST usada para implantar um cluster é processada de forma assíncrona e retorna um objeto Job.
- Objeto de erro: Se ocorrer um erro, um objeto de erro sempre será retornado. Por exemplo, você receberá um erro ao tentar criar um cluster com um nome que já existe.
- Vazio Em certos casos, nenhum dado é retornado e o corpo da resposta fica vazio. Por exemplo, o corpo da resposta fica vazio após usar DELETE para excluir um host existente.

## Processamento assíncrono usando o objeto Job para ONTAP Select

Algumas chamadas da API de implantação, especialmente aquelas que criam ou modificam um recurso, podem levar mais tempo para serem concluídas do que outras chamadas. O ONTAP Select Deploy processa essas solicitações de longa duração de forma assíncrona.

#### Solicitações assíncronas descritas usando o objeto Job

Após realizar uma chamada de API executada de forma assíncrona, o código de resposta HTTP 202 indica

que a solicitação foi validada e aceita com sucesso, mas ainda não foi concluída. A solicitação é processada como uma tarefa em segundo plano que continua em execução após a resposta HTTP inicial ao cliente. A resposta inclui o objeto Job que ancora a solicitação, incluindo seu identificador exclusivo.

Você deve consultar a página de documentação on-line do ONTAP Select Deploy para determinar quais chamadas de API operam de forma assíncrona.

**Consultar o objeto Job associado a uma solicitação de API**

O objeto Job retornado na resposta HTTP contém diversas propriedades. Você pode consultar a propriedade state para determinar se a solicitação foi concluída com sucesso. Um objeto Job pode estar em um dos seguintes estados:

- Na fila
- Correndo
- Sucesso
- Falha

Há duas técnicas que você pode usar ao pesquisar um objeto Job para detectar um estado terminal para a tarefa, seja sucesso ou falha:

- Solicitação de pesquisa padrão O estado do trabalho atual é retornado imediatamente
- Solicitação de pesquisa longa O estado do trabalho é retornado somente quando ocorre uma das seguintes situações:
  - O estado mudou mais recentemente do que o valor de data e hora fornecido na solicitação de pesquisa
  - O valor de tempo limite expirou (1 a 120 segundos)

A pesquisa padrão e a pesquisa longa usam a mesma chamada de API para consultar um objeto Job. No entanto, uma solicitação de pesquisa longa inclui dois parâmetros de consulta: `poll_timeout` e `last_modified`.

Você deve sempre usar long polling para reduzir a carga de trabalho na máquina virtual Deploy.

**Procedimento geral para emissão de uma solicitação assíncrona**

Você pode usar o seguinte procedimento de alto nível para concluir uma chamada de API assíncrona:

1. Emita a chamada de API assíncrona.
2. Receba uma resposta HTTP 202 indicando aceitação bem-sucedida da solicitação.
3. Extraia o identificador do objeto Job do corpo da resposta.
4. Dentro de um loop, execute o seguinte em cada ciclo:
   a. Obtenha o estado atual do trabalho com uma solicitação de pesquisa longa
   b. Se o trabalho estiver em um estado não terminal (na fila, em execução), execute o loop novamente.
5. Pare quando o trabalho atingir um estado terminal (sucesso, falha).

# Acesso com um navegador

## Antes de acessar a API ONTAP Select Deploy com um navegador

Há várias coisas que você deve saber antes de usar a página de documentação on-line do Deploy.

#### Plano de implantação

Se você pretende emitir chamadas de API como parte da execução de tarefas administrativas ou de implantação específicas, considere criar um plano de implantação. Esses planos podem ser formais ou informais e geralmente contêm seus objetivos e as chamadas de API a serem utilizadas. Consulte Processos de fluxo de trabalho usando a API REST de implantação para obter mais informações.

#### Exemplos JSON e definições de parâmetros

Cada chamada de API é descrita na página de documentação usando um formato consistente. O conteúdo inclui notas de implementação, parâmetros de consulta e códigos de status HTTP. Além disso, você pode exibir detalhes sobre o JSON usado com as solicitações e respostas da API da seguinte forma:

- Valor de Exemplo: Se você clicar em *Valor de Exemplo* em uma chamada de API, uma estrutura JSON típica para a chamada será exibida. Você pode modificar o exemplo conforme necessário e usá-lo como entrada para sua solicitação.
- Modelo Se você clicar em *Modelo*, uma lista completa dos parâmetros JSON será exibida, com uma descrição para cada parâmetro.

#### Cuidado ao emitir chamadas de API

Todas as operações de API que você realiza usando a página de documentação de Implantação são operações ativas. Tome cuidado para não criar, atualizar ou excluir configurações ou outros dados por engano.

## Acesse a página de documentação do ONTAP Select Deploy

Você deve acessar a página de documentação on-line do ONTAP Select Deploy para exibir a documentação da API, bem como para emitir manualmente uma chamada de API.

#### Antes de começar

Você deve ter o seguinte:

- Endereço IP ou nome de domínio da máquina virtual ONTAP Select Deploy
- Nome de usuário e senha do administrador

#### Passos

1. Digite a URL no seu navegador e pressione **Enter**:

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

2. Sign in usando o nome de usuário e a senha do administrador.

**Resultado**

A página da documentação do Deploy é exibida com as chamadas organizadas por categoria na parte inferior da página.

## Entenda e execute uma chamada de API ONTAP Select Deploy

Os detalhes de todas as chamadas de API são documentados e exibidos em um formato comum na página de documentação online do ONTAP Select Deploy. Ao compreender uma única chamada de API, você pode acessar e interpretar os detalhes de todas as chamadas de API.

**Antes de começar**

Você precisa estar conectado à página de documentação online do ONTAP Select Deploy. Você precisa ter o identificador exclusivo atribuído ao seu cluster ONTAP Select quando ele foi criado.

**Sobre esta tarefa**

Você pode recuperar as informações de configuração que descrevem um cluster ONTAP Select usando seu identificador exclusivo. Neste exemplo, todos os campos classificados como baratos são retornados. No entanto, como prática recomendada, você deve solicitar apenas os campos específicos necessários.

**Passos**

1. Na página principal, role até o final e clique em **Cluster**.
2. Clique em **GET /clusters/{cluster_id}** para exibir os detalhes da chamada de API usada para retornar informações sobre um cluster ONTAP Select .

# Processos de fluxo de trabalho

## Antes de usar os fluxos de trabalho da API ONTAP Select Deploy

Você deve se preparar para revisar e usar os processos de fluxo de trabalho.

**Entenda as chamadas de API usadas nos fluxos de trabalho**

A página de documentação online do ONTAP Select inclui os detalhes de cada chamada de API REST. Em vez de repetir esses detalhes aqui, cada chamada de API usada nos exemplos de fluxo de trabalho inclui apenas as informações necessárias para localizá-la na página de documentação. Após localizar uma chamada de API específica, você pode revisar os detalhes completos da chamada, incluindo os parâmetros de entrada, formatos de saída, códigos de status HTTP e tipo de processamento da solicitação.

As seguintes informações são incluídas para cada chamada de API em um fluxo de trabalho para ajudar a localizar a chamada na página de documentação:

- Categoria As chamadas de API são organizadas na página de documentação em áreas ou categorias funcionalmente relacionadas. Para localizar uma chamada de API específica, role até o final da página e clique na categoria de API aplicável.
- Verbo HTTP O verbo HTTP identifica a ação realizada em um recurso. Cada chamada de API é executada por meio de um único verbo HTTP.
- Caminho O caminho determina o recurso específico ao qual a ação se aplica como parte da execução de uma chamada. A string do caminho é anexada à URL principal para formar a URL completa que identifica o recurso.

#### Crie uma URL para acessar diretamente a API REST

Além da página de documentação do ONTAP Select , você também pode acessar a API REST de Implantação diretamente por meio de uma linguagem de programação como Python. Nesse caso, a URL principal é ligeiramente diferente da URL usada para acessar a página de documentação online. Ao acessar a API diretamente, você deve anexar /api à string de domínio e porta. Por exemplo: `http://deploy.mycompany.com/api`

## Fluxo de trabalho 1: criar um cluster de avaliação de nó único ONTAP Select no ESXi

Você pode implantar um cluster ONTAP Select de nó único em um host VMware ESXi gerenciado pelo vCenter. O cluster é criado com uma licença de avaliação.

O fluxo de trabalho de criação de cluster difere nas seguintes situações:

- O host ESXi não é gerenciado pelo vCenter (host autônomo)
- Vários nós ou hosts são usados dentro do cluster
- O cluster é implantado em um ambiente de produção com uma licença adquirida
- O hipervisor KVM é usado em vez do VMware ESXi

#### 1. Registre a credencial do servidor vCenter

Ao implantar em um host ESXi gerenciado por um servidor vCenter, você deve adicionar uma credencial antes de registrar o host. O utilitário de administração de implantação pode então usar a credencial para autenticar no vCenter.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Implantar | PUBLICAR | /segurança/credenciais |

**Cachos**

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

**Entrada JSON (etapa 01)**

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

**Tipo de processamento**

Assíncrono

**Saída**

- ID da credencial no cabeçalho de resposta de localização
- Objeto de trabalho

### 2. Registre um host do hipervisor

Você deve adicionar um host do hipervisor onde a máquina virtual que contém o nó ONTAP Select será executada.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | PUBLICAR | /anfitriões |

**Cachos**

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

**Entrada JSON (etapa 02)**

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

**Tipo de processamento**

Assíncrono

**Saída**

- ID do host no cabeçalho de resposta de localização
- Objeto de trabalho

### 3. Crie um cluster

Quando você cria um cluster ONTAP Select , a configuração básica do cluster é registrada e os nomes dos nós são gerados automaticamente pelo Deploy.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | PUBLICAR | /clusters |

**Cachos**

O parâmetro de consulta node_count deve ser definido como 1 para um cluster de nó único.

```
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 (etapa 03)**

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

**Tipo de processamento**

Síncrono

**Saída**

- ID do cluster no cabeçalho de resposta de localização

#### 4. Configurar o cluster

Há vários atributos que você deve fornecer como parte da configuração do cluster.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | CORREÇÃO | /clusters/{id_do_cluster} |

**Cachos**

Você deve fornecer o ID do cluster.

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

**Entrada JSON (etapa 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 processamento**

Síncrono

**Saída**

Nenhum

**5. Recupere o nome do nó**

O utilitário de administração Deploy gera automaticamente os identificadores e nomes dos nós quando um cluster é criado. Antes de configurar um nó, você precisa recuperar o ID atribuído.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | PEGAR | /clusters/{cluster_id}/nós |

**Cachos**

Você deve fornecer o ID do cluster.

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

**Tipo de processamento**

Síncrono

**Saída**

- A matriz registra cada um descrevendo um único nó com ID e nome exclusivos

**6. Configurar os nós**

Você deve fornecer a configuração básica para o nó, que é a primeira das três chamadas de API usadas para configurar um nó.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | CAMINHO | /clusters/{id_do_cluster}/nós/{id_do_nó} |

**Cachos**

Você deve fornecer o ID do cluster e o ID do nó.

```
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 (etapa 06)**

Você deve fornecer o ID do host onde o nó ONTAP Select será executado.

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

**Tipo de processamento**

Síncrono

**Saída**

Nenhum

### 7. Recupere as redes de nós

Você deve identificar as redes de dados e gerenciamento usadas pelo nó no cluster de nó único. A rede interna não é usada com um cluster de nó único.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | PEGAR | /clusters/{id_do_cluster}/nós/{id_do_nó}/redes |

**Cachos**

Você deve fornecer o ID do cluster e o ID do nó.

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

**Tipo de processamento**

Síncrono

**Saída**

- Matriz de dois registros, cada um descrevendo uma única rede para o nó, incluindo o ID exclusivo e a finalidade

### 8. Configurar a rede do nó

Você deve configurar as redes de dados e gerenciamento. A rede interna não é usada com um cluster de nó único.

Emita a seguinte chamada de API duas vezes, uma para cada rede.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | CORREÇÃO | /clusters/{id_do_cluster}/nós/{id_do_nó}/redes/{id_da_rede} |

**Cachos**

Você deve fornecer o ID do cluster, o ID do nó e o ID da rede.

```
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 (etapa 08)**

Você precisa fornecer o nome da rede.

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

**Tipo de processamento**

Síncrono

**Saída**

Nenhum

### 9. Configurar o pool de armazenamento de nós

A etapa final na configuração de um nó é anexar um pool de armazenamento. Você pode determinar os pools de armazenamento disponíveis por meio do cliente web do vSphere ou, opcionalmente, por meio da API REST de implantação.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | CORREÇÃO | /clusters/{id_do_cluster}/nós/{id_do_nó}/redes/{id_da_rede} |

**Cachos**

Você deve fornecer o ID do cluster, o ID do nó e o ID da rede.

```
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 (etapa 09)**

A capacidade do pool é de 2 TB.

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

**Tipo de processamento**

Síncrono

**Saída**

Nenhum

### 10. Implante o cluster

Depois que o cluster e o nó forem configurados, você poderá implantar o cluster.

| Categoria | verbo HTTP | Caminho |
|---|---|---|
| Conjunto | PUBLICAR | /clusters/{cluster_id}/implantar |

**Cachos**

Você deve fornecer o ID do cluster.

```
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 (etapa 10)**

Você deve fornecer a senha para a conta de administrador do ONTAP .

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

**Tipo de processamento**

Assíncrono

**Saída**

- Objeto de trabalho

**Informações relacionadas**

"Implantar uma instância de avaliação de 90 dias de um cluster ONTAP Select"

# Acesso com Python

## Antes de acessar o ONTAP Select Implantar API usando Python

Você deve preparar o ambiente antes de executar os scripts Python de exemplo.

Antes de executar os scripts Python, você deve certificar-se de que o ambiente esteja configurado corretamente:

- A versão mais recente aplicável do Python2 deve estar instalada. Os códigos de exemplo foram testados com Python2. Eles também devem ser portáveis para Python3, mas não foram testados quanto à compatibilidade.
- As bibliotecas Requests e urllib3 devem estar instaladas. Você pode usar o pip ou outra ferramenta de gerenciamento Python, conforme apropriado para o seu ambiente.
- A estação de trabalho cliente onde os scripts são executados deve ter acesso de rede à máquina virtual ONTAP Select Deploy.

Além disso, você deve ter as seguintes informações:

- Endereço IP da máquina virtual de implantação
- Nome de usuário e senha de uma conta de administrador do Deploy

## Entenda os scripts Python para ONTAP Select Deploy

Os scripts Python de exemplo permitem que você execute diversas tarefas diferentes. Você deve entender os scripts antes de usá-los em uma instância de implantação ativa.

### Características comuns de design

Os scripts foram projetados com as seguintes características comuns:

- Executar a partir da interface de linha de comando em uma máquina cliente. Você pode executar os scripts Python em qualquer máquina cliente configurada corretamente. Consulte *Antes de começar* para obter mais informações.
- Aceitar parâmetros de entrada da CLI Cada script é controlado na CLI por meio de parâmetros de entrada.
- Ler arquivo de entrada: Cada script lê um arquivo de entrada com base em sua finalidade. Ao criar ou excluir um cluster, você deve fornecer um arquivo de configuração JSON. Ao adicionar uma licença de nó, você deve fornecer um arquivo de licença válido.
- Use um módulo de suporte comum. O módulo de suporte comum *deploy_requests.py* contém uma única classe. Ele é importado e usado por cada um dos scripts.

### Criar um cluster

Você pode criar um cluster ONTAP Select usando o script cluster.py. Com base nos parâmetros da CLI e no conteúdo do arquivo de entrada JSON, você pode modificar o script para o seu ambiente de implantação da seguinte maneira:

- Hipervisor: você pode implantar no ESXI ou KVM (dependendo da versão de implantação). Ao implantar

no ESXi, o hipervisor pode ser gerenciado pelo vCenter ou pode ser um host autônomo.

- Tamanho do cluster Você pode implantar um cluster de nó único ou de vários nós.
- Licença de avaliação ou produção Você pode implantar um cluster com uma licença de avaliação ou adquirida para produção.

Os parâmetros de entrada da CLI para o script incluem:

- Nome do host ou endereço IP do servidor de implantação
- Senha para a conta de usuário administrador
- Nome do arquivo de configuração JSON
- Sinalizador detalhado para saída de mensagem

#### Adicionar uma licença de nó

Se optar por implantar um cluster de produção, você deverá adicionar uma licença para cada nó usando o script *add_license.py*. Você pode adicionar a licença antes ou depois de implantar o cluster.

Os parâmetros de entrada da CLI para o script incluem:

- Nome do host ou endereço IP do servidor de implantação
- Senha para a conta de usuário administrador
- Nome do arquivo de licença
- Nome de usuário ONTAP com privilégios para adicionar a licença
- Senha para o usuário ONTAP

#### Excluir um cluster

Você pode excluir um cluster ONTAP Select existente usando o script *delete_cluster.py*.

Os parâmetros de entrada da CLI para o script incluem:

- Nome do host ou endereço IP do servidor de implantação
- Senha para a conta de usuário administrador
- Nome do arquivo de configuração JSON

# Exemplos de código Python

## Script para criar um cluster ONTAP Select

Você pode usar o script a seguir para criar um cluster com base nos parâmetros definidos no script e em um arquivo de entrada JSON.

```
#!/usr/bin/env python
##-----------------------------------------------------------------
#
# File: cluster.py
#
```

```
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##------------------------------------------------------------------------

import traceback
import argparse
import json
import logging

from deploy_requests import DeployRequests


def add_vcenter_credentials(deploy, config):
    """ Add credentials for the vcenter if present in the config """
    log_debug_trace()

    vcenter = config.get('vcenter', None)
    if vcenter and not deploy.resource_exists('/security/credentials',
                                              'hostname', vcenter[
'hostname']):
        log_info("Registering vcenter {} credentials".format(vcenter[
'hostname']))
        data = {k: vcenter[k] for k in ['hostname', 'username', 'password
']}
        data['type'] = "vcenter"
        deploy.post('/security/credentials', data)


def add_standalone_host_credentials(deploy, config):
    """ Add credentials for standalone hosts if present in the config.
        Does nothing if the host credential already exists on the Deploy.
    """
    log_debug_trace()

    hosts = config.get('hosts', [])
    for host in hosts:
```

```
        # The presense of the 'password' will be used only for standalone
hosts.
        # If this host is managed by a vcenter, it should not have a host
'password' in the json.
        if 'password' in host and not deploy.resource_exists(
'/security/credentials',
                                                             'hostname',
host['name']):
            log_info("Registering host {} credentials".format(host['name
']))
            data = {'hostname': host['name'], 'type': 'host',
                    'username': host['username'], 'password': host[
'password']}
            deploy.post('/security/credentials', data)


def register_unkown_hosts(deploy, config):
    ''' Registers all hosts with the deploy server.
        The host details are read from the cluster config json file.

        This method will skip any hosts that are already registered.
        This method will exit the script if no hosts are found in the
config.
    '''
    log_debug_trace()

    data = {"hosts": []}
    if 'hosts' not in config or not config['hosts']:
        log_and_exit("The cluster config requires at least 1 entry in the
'hosts' list got {}".format(config))

    missing_host_cnt = 0
    for host in config['hosts']:
        if not deploy.resource_exists('/hosts', 'name', host['name']):
            missing_host_cnt += 1
            host_config = {"name": host['name'], "hypervisor_type": host[
'type']}
            if 'mgmt_server' in host:
                host_config["management_server"] = host['mgmt_server']
                log_info(
                   "Registering from vcenter {mgmt_server}".format(**
host))

            if 'password' in host and 'user' in host:
                host_config['credential'] = {
                    "password": host['password'], "username": host['user
```

```
']}

            log_info("Registering {type} host {name}".format(**host))
            data["hosts"].append(host_config)

    # only post /hosts if some missing hosts were found
    if missing_host_cnt:
        deploy.post('/hosts', data, wait_for_job=True)


def add_cluster_attributes(deploy, config):
    ''' POST a new cluster with all needed attribute values.
        Returns the cluster_id of the new config
    '''
    log_debug_trace()

    cluster_config = config['cluster']
    cluster_id = deploy.find_resource('/clusters', 'name', cluster_config
['name'])

    if not cluster_id:
        log_info("Creating cluster config named {name}".format(
**cluster_config))

        # Filter to only the valid attributes, ignores anything else in
the json
        data = {k: cluster_config[k] for k in [
            'name', 'ip', 'gateway', 'netmask', 'ontap_image_version',
'dns_info', 'ntp_servers']}

        num_nodes = len(config['nodes'])

        log_info("Cluster properties: {}".format(data))

        resp = deploy.post('/v3/clusters?node_count={}'.format(num_nodes),
data)
        cluster_id = resp.headers.get('Location').split('/')[-1]

    return cluster_id


def get_node_ids(deploy, cluster_id):
    ''' Get the the ids of the nodes in a cluster. Returns a list of
node_ids.'''
    log_debug_trace()

    response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
```

```
    node_ids = [node['id'] for node in response.json().get('records')]
    return node_ids


def add_node_attributes(deploy, cluster_id, node_id, node):
    ''' Set all the needed properties on a node '''
    log_debug_trace()

    log_info("Adding node '{}' properties".format(node_id))

    data = {k: node[k] for k in ['ip', 'serial_number', 'instance_type',
                                 'is_storage_efficiency_enabled'] if k in
node}
    # Optional: Set a serial_number
    if 'license' in node:
        data['license'] = {'id': node['license']}

    # Assign the host
    host_id = deploy.find_resource('/hosts', 'name', node['host_name'])
    if not host_id:
        log_and_exit("Host names must match in the 'hosts' array, and the
nodes.host_name property")

    data['host'] = {'id': host_id}

    # Set the correct raid_type
    is_hw_raid = not node['storage'].get('disks')  # The presence of a
list of disks indicates sw_raid
    data['passthrough_disks'] = not is_hw_raid

    # Optionally set a custom node name
    if 'name' in node:
        data['name'] = node['name']

    log_info("Node properties: {}".format(data))
    deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id),
data)


def add_node_networks(deploy, cluster_id, node_id, node):
    ''' Set the network information for a node '''
    log_debug_trace()

    log_info("Adding node '{}' network properties".format(node_id))

    num_nodes = deploy.get_num_records('/clusters/{}/nodes'.format
(cluster_id))
```

```
    for network in node['networks']:

        # single node clusters do not use the 'internal' network
        if num_nodes == 1 and network['purpose'] == 'internal':
            continue

        # Deduce the network id given the purpose for each entry
        network_id = deploy.find_resource('/clusters/{}/nodes/{}/networks
'.format(cluster_id, node_id),
                                          'purpose', network['purpose'])
        data = {"name": network['name']}
        if 'vlan' in network and network['vlan']:
            data['vlan_id'] = network['vlan']

        deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format(
cluster_id, node_id, network_id), data)


def add_node_storage(deploy, cluster_id, node_id, node):
    ''' Set all the storage information on a node '''
    log_debug_trace()

    log_info("Adding node '{}' storage properties".format(node_id))
    log_info("Node storage: {}".format(node['storage']['pools']))

    data = {'pool_array': node['storage']['pools']}  # use all the json
properties
    deploy.post(
        '/clusters/{}/nodes/{}/storage/pools'.format(cluster_id, node_id),
data)

    if 'disks' in node['storage'] and node['storage']['disks']:
        data = {'disks': node['storage']['disks']}
        deploy.post(
            '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id,
node_id), data)


def create_cluster_config(deploy, config):
    ''' Construct a cluster config in the deploy server using the input
json data '''
    log_debug_trace()

    cluster_id = add_cluster_attributes(deploy, config)

    node_ids = get_node_ids(deploy, cluster_id)
```

```
    node_configs = config['nodes']

    for node_id, node_config in zip(node_ids, node_configs):
        add_node_attributes(deploy, cluster_id, node_id, node_config)
        add_node_networks(deploy, cluster_id, node_id, node_config)
        add_node_storage(deploy, cluster_id, node_id, node_config)

    return cluster_id


def deploy_cluster(deploy, cluster_id, config):
    ''' Deploy the cluster config to create the ONTAP Select VMs. '''
    log_debug_trace()
    log_info("Deploying cluster: {}".format(cluster_id))

    data = {'ontap_credential': {'password': config['cluster'][
'ontap_admin_password']}}
    deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format
(cluster_id),
                data, wait_for_job=True)


def log_debug_trace():
    stack = traceback.extract_stack()
    parent_function = stack[-2][2]
    logging.getLogger('deploy').debug('Calling %s()' % parent_function)


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)


def configure_logging(verbose):
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    if verbose:
        logging.basicConfig(level=logging.DEBUG, format=FORMAT)
    else:
        logging.basicConfig(level=logging.INFO, format=FORMAT)
        logging.getLogger('requests.packages.urllib3.connectionpool'
).setLevel(
            logging.WARNING)
```

```
def main(args):
    configure_logging(args.verbose)
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)

        add_vcenter_credentials(deploy, config)

        add_standalone_host_credentials(deploy, config)

        register_unkown_hosts(deploy, config)

        cluster_id = create_cluster_config(deploy, config)

        deploy_cluster(deploy, cluster_id, config)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to construct and deploy a cluster.')
    parser.add_argument('-d', '--deploy', help='Hostname or IP address of
Deploy server')
    parser.add_argument('-p', '--password', help='Admin password of Deploy
server')
    parser.add_argument('-c', '--config_file', help='Filename of the
cluster config')
    parser.add_argument('-v', '--verbose', help='Display extra debugging
messages for seeing exact API calls and responses',
                        action='store_true', default=False)
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
    main(args)
```

## JSON para script para criar um cluster ONTAP Select

Ao criar ou excluir um cluster ONTAP Select usando os exemplos de código Python, você deve fornecer um arquivo JSON como entrada para o script. Você pode copiar e modificar o exemplo JSON apropriado de acordo com seus planos de implantação.

### Cluster de nó único no ESXi

```
{
```

```
  "hosts": [
    {
      "password": "mypassword1",
      "name": "host-1234",
      "type": "ESX",
      "username": "admin"
    }
  ],

  "cluster": {
    "dns_info": {
      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
        "lab3.company-demo.com", "lab4.company-demo.com"
        ],

      "dns_ips": ["10.206.80.135", "10.206.80.136"]
      },
      "ontap_image_version": "9.7",
      "gateway": "10.206.80.1",
      "ip": "10.206.80.115",
      "name": "mycluster",
      "ntp_servers": ["10.206.80.183", "10.206.80.142"],
      "ontap_admin_password": "mypassword2",
      "netmask": "255.255.254.0"
  },

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",
      "name": "node-1",
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
          "vlan": 1234
        },
        {
          "name": "ontap-external",
          "purpose": "data",
          "vlan": null
        },
        {
          "name": "ontap-internal",
          "purpose": "internal",
          "vlan": null
```

```
        }
      ],
      "host_name": "host-1234",
      "is_storage_efficiency_enabled": false,
      "instance_type": "small",
      "storage": {
        "disk": [],
        "pools": [
          {
            "name": "storage-pool-1",
            "capacity": 4802666790125
          }
        ]
      }
    }
  ]
}
```

**Cluster de nó único no 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
          }
        ]
      }
    }
  ]
}
```

### Cluster de nó único em 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 adicionar uma licença de nó ONTAP Select

Você pode usar o seguinte script para adicionar uma licença para um nó 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 excluir um cluster ONTAP Select

Você pode usar o seguinte script CLI para excluir um cluster 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 suporte comum para ONTAP Select

Todos os scripts Python usam uma classe Python comum em um 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 nós do cluster ONTAP Select

Você pode usar o seguinte script para redimensionar os nós em um cluster ONTAP Select .

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: resize_nodes.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import argparse
import logging
import sys

from deploy_requests import DeployRequests


def _parse_args():
    """ Parses the arguments provided on the command line when executing
this
        script and returns the resulting namespace. If all required
arguments
        are not provided, an error message indicating the mismatch is
printed and
        the script will exit.
    """

    parser = argparse.ArgumentParser(description=(
        'Uses the ONTAP Select Deploy API to resize the nodes in the
cluster.'
        ' For example, you might have a small (4 CPU, 16GB RAM per node) 2
node'
```

```
        ' cluster and wish to resize the cluster to medium (8 CPU, 64GB
RAM per'
        ' node). This script will take in the cluster details and then
perform'
        ' the operation and wait for it to complete.'
    ))
    parser.add_argument('--deploy', required=True, help=(
        'Hostname or IP of the ONTAP Select Deploy VM.'
    ))
    parser.add_argument('--deploy-password', required=True, help=(
        'The password for the ONTAP Select Deploy admin user.'
    ))
    parser.add_argument('--cluster', required=True, help=(
        'Hostname or IP of the cluster management interface.'
    ))
    parser.add_argument('--instance-type', required=True, help=(
        'The desired instance size of the nodes after the operation is
complete.'
    ))
    parser.add_argument('--ontap-password', required=True, help=(
        'The password for the ONTAP administrative user account.'
    ))
    parser.add_argument('--ontap-username', default='admin', help=(
        'The username for the ONTAP administrative user account. Default:
admin.'
    ))
    parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME', help=(
        'A space separated list of node names for which the resize
operation'
        ' should be performed. The default is to apply the resize to all
nodes in'
        ' the cluster. If a list of nodes is provided, it must be provided
in HA'
        ' pairs. That is, in a 4 node cluster, nodes 1 and 2 (partners)
must be'
        ' resized in the same operation.'
    ))
    return parser.parse_args()


def _get_cluster(deploy, parsed_args):
    """ Locate the cluster using the arguments provided """

    cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args
.cluster)
    if not cluster_id:
```

```
        return None
    return deploy.get('/clusters/%s?fields=nodes' % cluster_id).json()[
'record']


def _get_request_body(parsed_args, cluster):
    """ Build the request body """

    changes = {'admin_password': parsed_args.ontap_password}

    # if provided, use the list of nodes given, else use all the nodes in
the cluster
    nodes = [node for node in cluster['nodes']]
    if parsed_args.nodes:
        nodes = [node for node in nodes if node['name'] in parsed_args
.nodes]

    changes['nodes'] = [
        {'instance_type': parsed_args.instance_type, 'id': node['id']} for
node in nodes]

    return changes


def main():
    """ Set up the resize operation by gathering the necessary data and
then send
        the request to the ONTAP Select Deploy server.
    """

    logging.basicConfig(
        format='[%(asctime)s] [%(levelname)5s] %(message)s', level=
logging.INFO,)

    logging.getLogger('requests.packages.urllib3').setLevel(logging
.WARNING)

    parsed_args = _parse_args()
    deploy = DeployRequests(parsed_args.deploy, parsed_args
.deploy_password)

    cluster = _get_cluster(deploy, parsed_args)
    if not cluster:
        deploy.logger.error(
            'Unable to find a cluster with a management IP of %s' %
parsed_args.cluster)
        return 1
```

```
    changes = _get_request_body(parsed_args, cluster)
    deploy.patch('/clusters/%s' % cluster['id'], changes, wait_for_job
=True)

if __name__ == '__main__':
    sys.exit(main())
```

## Informações sobre direitos autorais



## Informações sobre marcas comerciais