



Automatize com REST

ONTAP Select

NetApp
January 31, 2025

Índice

Automatize com REST	1
Conceitos	1
Acesso com um navegador	8
Processos de fluxo de trabalho	10
Acesso com Python	17
Amostras de código Python	19

Automatize com REST

Conceitos

Base de serviços web REST

Representational State Transfer (REST) é um estilo para a criação de aplicações web distribuídas. Quando aplicada ao design de uma API de serviços da Web, ela estabelece um conjunto de tecnologias e melhores práticas para expor recursos baseados em servidor e gerenciar seus estados. Ele usa protocolos e padrões convencionais para fornecer uma base flexível para implantar e gerenciar clusters ONTAP Select.

Arquitetura e restrições clássicas

REST foi formalmente articulado por Roy Fielding em seu doutorado "[dissertação](#)" na UC Irvine em 2000. Ele define um estilo arquitetônico por meio de um conjunto de restrições, que coletivamente melhoraram as aplicações baseadas na Web e os protocolos subjacentes. As restrições estabelecem um aplicativo de serviços da Web RESTful com base em uma arquitetura cliente/servidor usando um protocolo de comunicação sem estado.

Recursos e representação do Estado

Os recursos são os componentes básicos de um sistema baseado na Web. Ao criar um aplicativo REST de serviços da Web, as tarefas iniciais de design incluem:

- Identificação de recursos baseados em sistema ou servidor que cada sistema usa e mantém recursos. Um recurso pode ser um arquivo, transação comercial, processo ou entidade administrativa. Uma das primeiras tarefas no projeto de um aplicativo baseado em serviços web REST é identificar os recursos.
- Definição de estados de recursos e recursos de operações estatais associadas 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 de extremidade URI

Todos os recursos REST devem ser definidos e disponibilizados usando um esquema de endereçamento bem definido. Os endpoints onde os recursos estão localizados e identificados usam um URI (Uniform Resource Identifier). O URI fornece uma estrutura geral para criar um nome exclusivo para cada recurso na rede. O Uniform Resource Locator (URL) é um tipo de URI usado com serviços da Web para identificar e acessar recursos. Os recursos são normalmente expostos em uma estrutura hierárquica semelhante a um diretório de arquivos.

Mensagens HTTP

O Hypertext Transfer Protocol (HTTP) é o protocolo usado pelo cliente e servidor de serviços da Web para trocar mensagens de solicitação e resposta sobre os recursos. Como parte do projeto de um aplicativo de serviços da Web, os verbos HTTP (como GET e POST) são mapeados para os recursos e ações de gerenciamento de estado correspondentes.

HTTP está sem estado. Portanto, para associar um conjunto de solicitações e respostas relacionadas em uma transação, informações adicionais devem ser incluídas nos cabeçalhos HTTP carregados 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 várias maneiras, a opção mais popular (e a usada com a API REST de implantação) é JavaScript Object Notation (JSON). JSON é um padrão da indústria para representar estruturas de dados simples em texto simples e é usado para transferir informações de estado descrevendo os recursos.

Como acessar a API Deploy

Devido à flexibilidade inerente dos serviços da Web REST, a API de implantação do ONTAP Select pode ser acessada de várias maneiras diferentes.

Implantar interface de usuário nativa do utilitário

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

Página de documentação on-line do ONTAP Select Deploy

A página de documentação on-line do ONTAP Select Deploy fornece um ponto de acesso alternativo ao usar um navegador. Além de fornecer uma maneira de executar chamadas individuais de API 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 várias áreas ou categorias funcionais diferentes.

Programa personalizado

Você pode acessar a API Deploy usando qualquer uma das várias linguagens e ferramentas de programação diferentes. As escolhas populares incluem Python, Java e curl. Um programa, script ou ferramenta que usa a API atua como um cliente de serviços da 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.

Implantar o controle de versão da API

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 de implantação. Você deve estar ciente da versão da API incluída na versão de implantação e como isso pode afetar o uso da API.

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

Implantar o 2,8 e posterior

O ONTAP Select Deploy 2,8 e todas as versões posteriores incluem a versão 3 da API REST.

Implantar o 2.7.2 e versões anteriores

O 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 implantar o 2,8 ou posterior a partir de uma versão anterior que inclua a versão 2 da API, você deve atualizar qualquer código existente que acesse diretamente a API, bem como quaisquer scripts usando a interface de linha de comando.

Caraterísticas operacionais básicas

Embora O REST estabeleça um conjunto comum de tecnologias e práticas recomendadas, os detalhes de cada API podem variar com base nas escolhas de design. Você deve estar ciente dos detalhes e das caraterísticas operacionais da API de implantação do ONTAP Select antes de usar a API.

Host de hipervisor versus nó ONTAP Select

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

Identificadores de objeto

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



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

Solicitar identificadores

Cada solicitação de API bem-sucedida é atribuído 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 evento de uma transação com base no ID de 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 um código de status 202. Isso indica que o servidor aceitou a solicitação do cliente e iniciou uma tarefa em segundo plano para concluir a solicitação. O sucesso ou falha final não está disponível imediatamente e deve ser determinado por meio de chamadas de API adicionais.

Confirme a conclusão de um trabalho de longa duração

Geralmente, qualquer operação que possa levar muito tempo para ser concluída é processada assincronamente usando uma tarefa em segundo plano no servidor. Com a API Deploy REST, 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 depois que uma tarefa em segundo plano é criada.

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

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

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

Segurança

A API Deploy 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 meio do TLS. O uso do protocolo HTTP em um canal não criptografado não é suportado. O TLS versão 1,2 é suportado.
- Autenticação HTTP a autenticação básica é usada para cada transação de API. Um cabeçalho HTTP, que inclui o nome de usuário e senha em uma cadeia de caracteres base64, é adicionado a cada solicitação.

Transação de API de solicitação e resposta

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

- Se o corpo da solicitação incluir JSON, esse cabeçalho deve ser definido como `application/json`.
- Aceitar se o corpo da resposta incluir JSON, esse cabeçalho deve ser definido como `application/json`.
- A autenticação básica de autorização deve ser definida com o nome de usuário e senha codificados em uma cadeia de caracteres base64.

Corpo do pedido

O conteúdo do corpo da solicitação varia de acordo com a 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:

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

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

Operador	Descrição
.	Igual a.
*	Menos de
>	Superior a.
O que é que eu tenho	Inferior ou igual a
>	Maior ou igual a
	Ou
!	Não é igual a
*	Wildcard ganancioso

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

Selecionar campos de objeto

Por padrão, a emissão de uma chamada de API usando O GET retorna apenas os atributos que identificam exclusivamente o objeto ou objetos. Este conjunto mínimo de campos atua como uma chave para cada objeto e varia de acordo com o tipo de objeto. Você pode selecionar propriedades de objeto adicionais usando o parâmetro de consulta campos das seguintes maneiras:

- Campos de baixo custo especificam `fields=*` para recuperar os campos de objeto que são mantidos na memória do servidor local ou requerem pouco processamento para acessar.
- Campos caros especificam `fields=**` para recuperar todos os campos de objeto, incluindo aqueles que exigem processamento adicional de servidor para acessar.
- Seleção de campo personalizada Use `fields=FIELDNAME` para especificar o campo exato desejado. Ao solicitar vários campos, os valores devem ser separados usando vírgulas sem espaços.



Como prática recomendada, você deve sempre identificar os campos específicos que deseja. Você só deve recuperar o conjunto de campos baratos ou caros quando necessário. A classificação barata e cara é determinada pelo NetApp com base na análise interna 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 ordenação da seguinte forma:

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

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

```
order_by=tipo 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` com a 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 percorrer os 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 da API

Cada solicitação de API gera uma resposta de volta ao cliente. Você pode examinar a resposta para determinar se ela foi bem-sucedida e recuperar dados adicionais, conforme 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 é criado com sucesso; o cabeçalho de resposta de localização inclui o identificador exclusivo para o objeto.
202	Aceito	Foi iniciado um trabalho em segundo plano de longa execução para executar a solicitação, mas a operação ainda não foi concluída.
400	Pedido incorreto	A entrada de solicitação não é reconhecida ou é inadequada.
403	Proibido	O acesso é negado devido a um erro de autorização.

Código	Significado	Descrição
404	Não encontrado	O recurso referido 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 estão incluídos na resposta HTTP gerada pelo servidor de implantação, incluindo:

- Cada solicitação de API bem-sucedida é atribuída a um identificador de solicitação exclusivo.
- Localização quando um objeto é criado, o cabeçalho do local inclui o URL completo para o novo objeto, incluindo o identificador de objeto exclusivo.

Corpo de resposta

O conteúdo da resposta associada a uma solicitação de API difere com base no objeto, no tipo de processamento e no sucesso ou falha da solicitação. O corpo de resposta é renderizado em JSON.

- 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 de uma coleção de recursos podem ser retornados. Em todos os casos, há um formato consistente usado, com `num_records` a indicação do número de Registros e Registros contendo um array das instâncias do objeto. Por exemplo, você pode recuperar todos os nós definidos em um cluster específico.
- Objeto de tarefa se uma chamada de API for processada de forma assíncrona, um objeto Job será retornado que ancora 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.
- Se ocorrer um erro, um objeto de erro é sempre 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 de resposta está vazio. Por exemplo, o corpo da resposta está vazio depois de usar DELETE para excluir um host existente.

Processamento assíncrono usando o objeto de tarefa

Algumas das chamadas da API Deploy, particularmente 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

Depois de fazer uma chamada de API que é 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 a ser executada após a resposta HTTP inicial

ao cliente. A resposta inclui o objeto Job ancorando 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 assincronamente.

Consulte o objeto Job associado a uma solicitação de API

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

- Em fila de espera
- Em execução
- Sucesso
- Falha

Há duas técnicas que você pode usar ao fazer polling de um objeto Job para detetar um estado terminal para a tarefa, seja com sucesso ou falha:

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

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



Você deve sempre usar polling longo para reduzir a carga de trabalho na máquina virtual de implantação.

Procedimento geral para emitir uma solicitação assíncrona

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

1. Emita a chamada assíncrona da API.
2. Receber uma resposta HTTP 202 indicando aceitação bem-sucedida da solicitação.
3. Extraia o identificador do objeto Job do corpo de resposta.
4. Dentro de um loop, execute o seguinte em cada ciclo:
 - a. Obtenha o estado atual do trabalho com uma solicitação de poll longa
 - b. Se o trabalho estiver em um estado não terminal (em fila de espera, 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 com um navegador

Há várias coisas que você deve estar ciente antes de usar a página implantar documentação on-line.

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 usadas. 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:

- Exemplo de valor se você clicar em *exemplo de valor* 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 *Model*, 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 executa usando a página de documentação de implantação são operações ativas. Você deve ter cuidado para não criar, atualizar ou excluir configurações ou outros dados por engano.

Acesse a página implantar documentação

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 utilizador e palavra-passe do administrador

Passos

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

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

2. Inicie sessão utilizando o nome de utilizador e a palavra-passe do administrador.

Resultado

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

Compreender e executar uma chamada de API

Os detalhes de todas as chamadas de API são documentados e exibidos usando um formato comum na página da Web de documentação on-line do ONTAP Select Deploy. Ao entender uma única chamada de API, você pode acessar e interpretar os detalhes de todas as chamadas de API.

Antes de começar

Você deve estar conectado à página da Web de documentação on-line do ONTAP Select Deploy. Você deve ter o identificador exclusivo atribuído ao cluster do ONTAP Select quando o cluster 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 devolvidos. No entanto, como prática recomendada, você deve solicitar apenas os campos específicos necessários.

Passos

1. Na página principal, role até a parte inferior 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

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 on-line 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 localizar a chamada na página de documentação. Depois de 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 de solicitação.

As seguintes informações são incluídas para cada chamada de API dentro de 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 a parte inferior da página e clique na categoria de API aplicável.
- HTTP verb o verbo HTTP identifica a ação executada em um recurso. Cada chamada de API é executada através 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 cadeia de caracteres do caminho é anexada ao URL principal para formar a URL completa que identifica o recurso.

Crie um 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. Neste caso, o URL principal é ligeiramente diferente do URL usado ao acessar a página de documentação on-line. Ao acessar a API diretamente, você deve anexar /api ao domínio e à cadeia de caracteres da porta. Por exemplo:

```
http://deploy.mycompany.com/api
```

Fluxo de trabalho 1: Crie um cluster de avaliação de nó único 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 vCenter Server

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 implantar pode usar a credencial para autenticar no vCenter.

Categoria	Verbo HTTP	Caminho
Implantar	POST	/security/credentials

Curl

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

Entrada JSON (step01)

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

Tipo de processamento

Assíncrono

Saída

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

2. Registre um host de hipervisor

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

Categoria	Verbo HTTP	Caminho
Cluster	POST	/hosts

Curl

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

Entrada JSON (step02)

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

Tipo de processamento

Assíncrono

Saída

- ID do host no cabeçalho da resposta do local
- Objeto 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 pela implantação.

Categoria	Verbo HTTP	Caminho
Cluster	POST	/clusters

Curl

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 (step03)

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

Tipo de processamento

Síncrono

Saída

- ID do cluster no cabeçalho de resposta do local

4. Configure o cluster

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

Categoria	Verbo HTTP	Caminho
Cluster	PATCH	[cluster_id]

Curl

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 (step04)

```
{
  "dns_info": {
    "domains": ["lab1.company-demo.com"],
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
  },
  "ontap_image_version": "9.5",
  "gateway": "10.206.80.1",
  "ip": "10.206.80.115",
  "netmask": "255.255.255.192",
  "ntp_servers": {"10.206.80.183"}
}
```

Tipo de processamento

Síncrono

Saída

Nenhum

5. Recupere o nome do nó

O utilitário de administração implantar gera automaticamente os identificadores e nomes dos nós quando um cluster é criado. Antes de poder configurar um nó, tem de recuperar a ID atribuída.

Categoria	Verbo HTTP	Caminho
Cluster	OBTER	/clusters/_cluster_id/nós

Curl

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

- Array Registra cada um descrevendo um único nó com o ID e o nome exclusivos

6. Configure os nós

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

Categoria	Verbo HTTP	Caminho
Cluster	CAMINHO	/clusters/cluster_id/node/node_id

Curl

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 (step06)

Você deve fornecer a 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 os dados e as redes de 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
Cluster	OBTER	/clusters/cluster_id/nodes/node_id/networks

Curl

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

- Array de dois Registros cada um descrevendo uma única rede para o nó, incluindo a ID e a finalidade exclusivos

8. Configure a rede do nó

Você deve configurar os dados e as redes de 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
Cluster	PATCH	/clusters/cluster_id/node/node_id/networks/ network_id

Curl

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 (step08)

Você precisa fornecer o nome da rede.

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

Tipo de processamento

Síncrono

Saída

Nenhum

9. Configure o pool de storage de nós

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

Categoria	Verbo HTTP	Caminho
Cluster	PATCH	/clusters/cluster_id/node/node_id/networks/ network_id

Curl

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 (step09)

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 tiverem sido configurados, você poderá implantar o cluster.

Categoria	Verbo HTTP	Caminho
Cluster	POST	/clusters/cluster_id/deploy

Curl

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 (step10)

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

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

Tipo de processamento

Assíncrono

Saída

- Objeto trabalho

Acesso com Python

Antes de acessar a 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 está configurado corretamente:

- A versão aplicável mais recente do python2 deve ser instalada. Os códigos de amostra foram testados usando python2. Eles também devem ser portáteis para Python3, mas não foram testados para compatibilidade.
- As solicitações e bibliotecas urllib3 devem ser instaladas. Você pode usar o pip ou outra ferramenta de gerenciamento Python conforme apropriado para o seu ambiente.
- A estação de trabalho cliente em que os scripts são executados deve ter acesso à 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 de implantação

Entenda os scripts Python

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

Caraterísticas comuns do projeto

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 a partir de qualquer máquina cliente configurada corretamente. Consulte *antes de começar* para obter mais informações.
- Aceitar parâmetros de entrada CLI cada script é controlado na CLI através de parâmetros de entrada.
- Ler arquivo de entrada cada script lê um arquivo de entrada com base em seu propósito. 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.

Crie um cluster

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

- Hypervisor você pode implantar no ESXi ou KVM (dependendo da versão de implantação). Ao implantar no ESXi, o hypervisor 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.
- Avaliação ou licença de 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
- Palavra-passe para a conta de utilizador admin
- Nome do arquivo de configuração JSON
- Sinalizador verboso para saída de mensagem

Adicione uma licença de nó

Se você optar por implantar um cluster de produção, será necessário 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
- Palavra-passe para a conta de utilizador admin
- Nome do ficheiro de licença
- Nome de usuário do ONTAP com Privileges para adicionar a licença
- Senha para o usuário do ONTAP

Eliminar 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
- Palavra-passe para a conta de utilizador admin
- Nome do arquivo de configuração JSON

Amostras de código Python

Script para criar um cluster

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

Ao criar ou excluir um cluster do ONTAP Select usando as amostras de código Python, você deve fornecer um arquivo JSON como entrada para o script. Você pode copiar e modificar a amostra JSON apropriada com base em 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 o 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 no 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

Você pode usar o script a seguir 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

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/{id}?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/{id}'.format(cluster_id), {'availability':
'powered_off'}, True)

def delete_cluster(deploy, cluster_id):
    log_info("Deleting the cluster({}).".format(cluster_id))
    deploy.delete('/clusters/{id}'.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 suporte comum

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/{id}?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 de cluster

Você pode usar o script a seguir para redimensionar os nós em um cluster do 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

Copyright © 2025 NetApp, Inc. Todos os direitos reservados. Impresso nos EUA. Nenhuma parte deste documento protegida por direitos autorais pode ser reproduzida de qualquer forma ou por qualquer meio — gráfico, eletrônico ou mecânico, incluindo fotocópia, gravação, gravação em fita ou storage em um sistema de recuperação eletrônica — sem permissão prévia, por escrito, do proprietário dos direitos autorais.

O software derivado do material da NetApp protegido por direitos autorais está sujeito à seguinte licença e isenção de responsabilidade:

ESTE SOFTWARE É FORNECIDO PELA NETAPP "NO PRESENTE ESTADO" E SEM QUAISQUER GARANTIAS EXPRESSAS OU IMPLÍCITAS, INCLUINDO, SEM LIMITAÇÕES, GARANTIAS IMPLÍCITAS DE COMERCIALIZAÇÃO E ADEQUAÇÃO A UM DETERMINADO PROPÓSITO, CONFORME A ISENÇÃO DE RESPONSABILIDADE DESTES DOCUMENTOS. EM HIPÓTESE ALGUMA A NETAPP SERÁ RESPONSÁVEL POR QUALQUER DANO DIRETO, INDIRETO, INCIDENTAL, ESPECIAL, EXEMPLAR OU CONSEQUENCIAL (INCLUINDO, SEM LIMITAÇÕES, AQUISIÇÃO DE PRODUTOS OU SERVIÇOS SOBRESSALIENTES; PERDA DE USO, DADOS OU LUCROS; OU INTERRUPÇÃO DOS NEGÓCIOS), INDEPENDENTEMENTE DA CAUSA E DO PRINCÍPIO DE RESPONSABILIDADE, SEJA EM CONTRATO, POR RESPONSABILIDADE OBJETIVA OU PREJUÍZO (INCLUINDO NEGLIGÊNCIA OU DE OUTRO MODO), RESULTANTE DO USO DESTES SOFTWARES, MESMO SE ADVERTIDA DA RESPONSABILIDADE DE TAL DANO.

A NetApp reserva-se o direito de alterar quaisquer produtos descritos neste documento, a qualquer momento e sem aviso. A NetApp não assume nenhuma responsabilidade nem obrigação decorrentes do uso dos produtos descritos neste documento, exceto conforme expressamente acordado por escrito pela NetApp. O uso ou a compra deste produto não representam uma licença sob quaisquer direitos de patente, direitos de marca comercial ou quaisquer outros direitos de propriedade intelectual da NetApp.

O produto descrito neste manual pode estar protegido por uma ou mais patentes dos EUA, patentes estrangeiras ou pedidos pendentes.

LEGENDA DE DIREITOS LIMITADOS: o uso, a duplicação ou a divulgação pelo governo estão sujeitos a restrições conforme estabelecido no subparágrafo (b)(3) dos Direitos em Dados Técnicos - Itens Não Comerciais no DFARS 252.227-7013 (fevereiro de 2014) e no FAR 52.227- 19 (dezembro de 2007).

Os dados aqui contidos pertencem a um produto comercial e/ou serviço comercial (conforme definido no FAR 2.101) e são de propriedade da NetApp, Inc. Todos os dados técnicos e software de computador da NetApp fornecidos sob este Contrato são de natureza comercial e desenvolvidos exclusivamente com despesas privadas. O Governo dos EUA tem uma licença mundial limitada, irrevogável, não exclusiva, intransferível e não sublicenciável para usar os Dados que estão relacionados apenas com o suporte e para cumprir os contratos governamentais desse país que determinam o fornecimento de tais Dados. Salvo disposição em contrário no presente documento, não é permitido usar, divulgar, reproduzir, modificar, executar ou exibir os dados sem a aprovação prévia por escrito da NetApp, Inc. Os direitos de licença pertencentes ao governo dos Estados Unidos para o Departamento de Defesa estão limitados aos direitos identificados na cláusula 252.227-7015(b) (fevereiro de 2014) do DFARS.

Informações sobre marcas comerciais

NETAPP, o logotipo NETAPP e as marcas listadas em <http://www.netapp.com/TM> são marcas comerciais da NetApp, Inc. Outros nomes de produtos e empresas podem ser marcas comerciais de seus respectivos proprietários.