



使用 **REST** 实现自动化 ONTAP Select

NetApp
January 29, 2026

目录

使用 REST 实现自动化	1
概念	1
用于部署和管理ONTAP Select集群的 REST Web 服务基础	1
如何访问ONTAP Select Deploy API	2
ONTAP Select Deploy API 版本控制	2
ONTAP Select Deploy API 基本操作特性	2
ONTAP Select的请求和响应 API 事务	4
使用ONTAP Select 的作业对象进行异步处理	6
使用浏览器访问	7
在使用浏览器访问ONTAP Select Deploy API 之前	7
访问ONTAP Select Deploy 文档页面	8
了解并执行ONTAP Select Deploy API 调用	8
工作流程	9
使用ONTAP Select Deploy API 工作流之前	9
工作流程 1: 在 ESXi 上创建ONTAP Select单节点评估集群	9
使用 Python 访问	16
在使用 Python 访问ONTAP Select Deploy API 之前	16
了解ONTAP Select Deploy 的 Python 脚本	16
Python 代码示例	18
用于创建ONTAP Select集群的脚本	18
用于创建ONTAP Select集群的脚本的 JSON	25
用于添加ONTAP Select节点许可证的脚本	29
用于删除ONTAP Select集群的脚本	33
ONTAP Select的通用支持 Python 模块	35
用于调整ONTAP Select集群节点大小的脚本	39

使用 REST 实现自动化

概念

用于部署和管理ONTAP Select集群的 REST Web 服务基础

表述性状态传输 (REST) 是一种用于创建分布式 Web 应用程序的样式。当应用于 Web 服务 API 的设计时，它可以建立一套用于公开基于服务器的资源并管理其状态的技术和最佳实践。它使用主流协议和标准，为部署和管理ONTAP Select集群提供灵活的基础。

建筑与经典约束

REST 由 Roy Fielding 在他的博士论文中正式提出 "论文" 2000年在加州大学欧文分校获得。它通过一系列约束定义了一种架构风格，这些约束共同改进了基于Web的应用程序及其底层协议。这些约束基于客户端/服务器架构，使用无状态通信协议，构建了一个RESTful Web服务应用程序。

资源和国家代表

资源是基于 Web 系统的基本组件。创建 REST Web 服务应用程序时，早期设计任务包括：

- 识别系统或基于服务器的资源。每个系统都会使用和维护资源。资源可以是文件、业务事务、流程或管理实体。基于 REST Web 服务设计应用程序的首要任务之一就是识别资源。
- 资源状态及相关状态操作的定义：资源始终处于有限数量的状态之一。必须明确定义这些状态以及用于影响状态变化的相关操作。

客户端和服务器之间交换消息，根据通用 CRUD（创建、读取、更新和删除）模型访问和更改资源的状态。

URI 端点

每个 REST 资源都必须使用定义明确的寻址方案进行定义和提供。资源所在和标识的端点使用统一资源标识符 (URI)。URI提供了一个通用框架，用于为网络中的每个资源创建唯一名称。统一资源定位符 (URL) 是一种用于 Web 服务识别和访问资源的 URI。资源通常以类似于文件目录的层级结构公开。

HTTP 消息

超文本传输协议 (HTTP) 是 Web 服务客户端和服务器用来交换资源请求和响应消息的协议。在设计 Web 服务应用程序的过程中，HTTP 动词（例如 GET 和 POST）会被映射到资源及其对应的状态管理操作。

HTTP 是无状态的。因此，为了将一组相关的请求和响应关联到一个事务下，必须在请求/响应数据流携带的 HTTP 标头中包含附加信息。

JSON 格式

虽然信息可以通过多种方式在客户端和服务器之间构建和传输，但最流行的选项（以及与部署 REST API 一起使用的选项）是 JavaScript 对象表示法 (JSON)。JSON是一种行业标准，用于以纯文本表示简单数据结构，并用于传输描述资源的状态信息。

如何访问ONTAP Select Deploy API

由于 REST Web 服务固有的灵活性，可以通过多种不同的方式访问ONTAP Select Deploy API。

部署实用程序原生用户界面

访问 API 的主要方式是通过ONTAP Select Deploy Web 用户界面。浏览器会调用 API 并根据用户界面的设计重新格式化数据。您也可以通过 Deploy 实用程序命令行界面访问 API。

ONTAP Select Deploy 在线文档页面

ONTAP Select Deploy 在线文档页面提供了使用浏览器时的备用访问点。除了提供直接执行各个 API 调用的方法外，该页面还包含 API 的详细说明，包括每个调用的输入参数和其他选项。API调用分为几个不同的功能区域或类别。

定制程序

您可以使用多种不同的编程语言和工具访问 Deploy API。常用的选择包括 Python、Java 和 cURL。使用该 API 的程序、脚本或工具可充当 REST Web 服务客户端。使用编程语言可以帮助您更好地理解 API，并实现ONTAP Select部署的自动化。

ONTAP Select Deploy API 版本控制

ONTAP Select Deploy 附带的 REST API 已分配一个版本号。API版本号与 Deploy 版本号无关。您应该了解您的 Deploy 版本中包含的 API 版本，以及这可能会对您使用该 API 产生的影响。

Deploy 管理实用程序的当前版本包含 REST API 版本 3。Deploy实用程序的先前版本包含以下 API 版本：

部署 2.8 及更高版本

ONTAP Select Deploy 2.8 及所有更高版本均包含 REST API 版本 3。

部署 2.7.2 及更早版本

ONTAP Select Deploy 2.7.2 和所有早期版本均包含 REST API 版本 2。



REST API 的版本 2 和 3 不兼容。如果您从包含 API 版本 2 的早期版本升级到 Deploy 2.8 或更高版本，则必须更新所有直接访问 API 的现有代码以及所有使用命令行界面的脚本。

ONTAP Select Deploy API 基本操作特性

虽然 REST 确立了一套通用的技术和最佳实践，但每个 API 的细节可能会因设计选择而异。在使用ONTAP Select Deploy API 之前，您应该了解该 API 的详细信息和操作特性。

虚拟机管理程序主机与ONTAP Select节点

虚拟机管理程序主机是托管ONTAP Select虚拟机的核心硬件平台。当ONTAP Select虚拟机部署在虚拟机管理程序主机上并处于活动状态时，该虚拟机将被视为 ONTAP Select 节点。在 Deploy REST API 版本 3 中，主机对象和节点对象是独立且不同的。这允许一对多关系，其中一个或多个ONTAP Select节点可以在同一虚拟机管理

程序主机上运行。

对象标识符

每个资源实例或对象在创建时都会分配一个唯一标识符。这些标识符在 ONTAP Select Deploy 的特定实例中是全局唯一的。发出创建新对象实例的 API 调用后，关联的 ID 值将返回给调用者 `location` HTTP 响应的标头。您可以提取标识符，并在后续调用中引用资源实例时使用它。



对象标识符的内容和内部结构可能随时更改。在引用关联对象时，应仅在适用的 API 调用中根据需要使用这些标识符。

请求标识符

每个成功的 API 请求都会被分配一个唯一的标识符。该标识符在 ``request-id`` 关联 HTTP 响应的标头。您可以使用请求标识符来统称单个特定 API 请求-响应事务的活动。例如，您可以根据请求 ID 检索事务的所有事件消息。

同步和异步调用

服务器执行从客户端收到的 HTTP 请求主要有两种方式：

- 同步服务器立即执行请求并以状态代码 200、201 或 204 进行响应。
- 异步：服务器接受请求并以状态码 202 进行响应。这表示服务器已接受客户端请求并启动后台任务以完成该请求。最终的成功或失败情况无法立即确定，必须通过其他 API 调用来确定。

确认长期运行作业的完成

通常，任何需要很长时间才能完成的操作都会使用服务器上的后台任务异步处理。使用 Deploy REST API，每个后台任务都由一个 Job 对象固定，该对象跟踪任务并提供信息（例如当前状态）。创建后台任务后，HTTP 响应中会返回一个 Job 对象（包含其唯一标识符）。

您可以直接查询 Job 对象来确定相关 API 调用是否成功。有关更多信息，请参阅 [使用 Job 对象进行异步处理](#)。

除了使用 Job 对象之外，还有其他方法可以确定请求的成功或失败，包括：

- 事件消息您可以使用原始响应返回的请求 ID 检索与特定 API 调用关联的所有事件消息。事件消息通常包含成功或失败的指示，在调试错误情况时也很有用。
- 资源状态或状况 一些资源维持一个状态或状况值，您可以查询该状态或状况值来间接确定请求的成功或失败。

安全性

Deploy API 使用以下安全技术：

- 传输层安全性 (TLS)：Deploy 服务器和客户端之间通过网络发送的所有流量均通过 TLS 加密。不支持在未加密的通道上使用 HTTP 协议。支持 TLS 1.2 版本。
- HTTP 身份验证 每个 API 事务都使用基本身份验证。每个请求都会添加一个 HTTP 标头，其中包含以 base64 字符串形式表示的用户名和密码。

ONTAP Select的请求和响应 API 事务

每个 Deploy API 调用都会以 HTTP 请求的形式发送到 Deploy 虚拟机，并生成一个相关的响应返回给客户端。此请求/响应对被视为一次 API 事务。在使用 Deploy API 之前，您应该熟悉可用于控制请求的输入变量以及响应输出的内容。

控制 API 请求的输入变量

您可以通过 HTTP 请求中设置的参数来控制如何处理 API 调用。

请求标头

您必须在 HTTP 请求中包含几个标头，其中包括：

- content-type 如果请求主体包含 JSON，则此标头必须设置为 application/json。
- accept 如果响应主体将包含 JSON，则此标头必须设置为 application/json。
- 授权必须使用以 base64 字符串编码的用户名和密码设置基本身份验证。

请求正文

请求主体的内容根据具体调用而有所不同。HTTP 请求主体由以下之一组成：

- 带有输入变量的 JSON 对象（例如，新集群的名称）
- 空

过滤对象

发出使用 GET 的 API 调用时，您可以根据任意属性限制或过滤返回的对象。例如，您可以指定要匹配的精确值：

<field>=<query value>

除了完全匹配之外，还有其他运算符可用于返回一组包含特定值范围的对象。ONTAPONTAP Select支持以下筛选运算符。

运算符	描述
=	等于
<	少于
>	大于
≤	小于或等于
≥	大于或等于
	或
!	不等于
*	贪婪通配符

您还可以通过使用 null 关键字或其否定 (! null) 作为查询的一部分，根据特定字段是否设置来返回一组对象。

选择对象字段

默认情况下，使用 GET 发出 API 调用仅返回唯一标识一个或多个对象的属性。这组最小字段将充当每个对象的键，并根据对象类型而有所不同。您可以使用 fields 查询参数通过以下方式选择其他对象属性：

- 廉价字段指定 `fields=`` 检索在本地服务器内存中维护或几乎不需要处理即可访问的对象字段。
- 昂贵字段指定 `fields=\*\*` 检索所有对象字段，包括那些需要额外的服务器处理才能访问的字段。
- 自定义字段选择使用 `fields=FIELDNAME` 指定您想要的确切字段。请求多个字段时，必须使用逗号分隔值，且不能使用空格。



作为最佳实践，您应该始终识别所需的特定字段。您应仅在需要时检索廉价或昂贵字段集。廉价和昂贵的分类由NetApp根据内部性能分析确定。给定字段的分类可能随时更改。

对输出集中的对象进行排序

资源集中的记录将按照对象定义的默认顺序返回。您可以使用 order_by 查询参数，并附带字段名称和排序方向来更改顺序，如下所示：

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

例如，您可以按降序对类型字段进行排序，然后按升序对 id 进行排序：

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

当包含多个参数时，必须用逗号分隔字段。

分页

使用 GET 方式发出 API 调用来访问相同类型的对象集合时，默认返回所有匹配的对象。如有需要，您可以使用请求中的 max_records 查询参数来限制返回的记录数。例如：

```
max_records=20
```

如果需要，您可以将此参数与其他查询参数组合使用，以缩小结果集。例如，以下内容返回在指定时间之后生成的最多 10 个系统事件：

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

您可以发出多个请求来分页浏览事件（或任何对象类型）。每个后续 API 调用都应根据最后一个结果集中的最新事件使用新的时间值。

解释 API 响应

每个 API 请求都会生成一个响应返回给客户端。您可以检查该响应以确定请求是否成功，并根据需要检索其他数据。

HTTP 状态代码

下面描述了部署 REST API 使用的 HTTP 状态代码。

代码	含义	描述
200	确定	表示没有创建新对象的调用成功。
201	已创建	对象已成功创建；位置响应标头包含该对象的唯一标识符。

代码	含义	描述
202	已接受	已启动长时间运行的后台作业来执行请求，但操作尚未完成。
400	错误的请求	请求输入无法识别或不合适。
403	禁止	由于授权错误，访问被拒绝。
404	未找到	请求中引用的资源不存在。
405	方法不允许	该资源不支持请求中的 HTTP 动词。
409	冲突	尝试创建对象失败，因为该对象已存在。
500	内部错误	服务器发生一般内部错误。
501	未实施	URI 已知但无法执行请求。

响应标头

Deploy 服务器生成的 HTTP 响应中包含几个标头，包括：

- request-id 每个成功的 API 请求都会分配一个唯一的请求标识符。
- 位置 当创建一个对象时，位置标头包含新对象的完整 URL，其中包括唯一对象标识符。

响应正文

与 API 请求相关的响应内容会根据对象、处理类型以及请求的成功或失败而有所不同。响应主体以 JSON 格式呈现。

- 单个对象：可以根据请求返回包含一组字段的单个对象。例如，您可以使用 GET 操作，通过唯一标识符检索集群的选定属性。
- 多个对象 可以返回资源集中的多个对象。在所有情况下，都使用一致的格式，`num\_records` 指示记录数以及包含对象实例数组的记录。例如，您可以检索特定集群中定义的所有节点。
- Job 对象如果 API 调用是异步处理的，则会返回一个 Job 对象，该对象用于锚定后台任务。例如，用于部署集群的 POST 请求是异步处理的，并返回一个 Job 对象。
- 错误对象 如果发生错误，始终会返回一个 Error 对象。例如，当您尝试创建一个名称已存在的集群时，就会收到错误。
- 空 某些情况下，没有返回任何数据，响应体为空，例如使用 DELETE 删除一个已经存在的主机，响应体为空。

使用 ONTAP Select 的作业对象进行异步处理

某些 Deploy API 调用（尤其是创建或修改资源的调用）可能需要比其他调用更长的时间才能完成。ONTAP Select Deploy 会异步处理这些长时间运行的请求。

使用 Job 对象描述的异步请求

进行异步运行的 API 调用后，HTTP 响应代码 202 表示请求已成功验证并接受，但尚未完成。该请求将作为后台任务处理，并在客户端收到初始 HTTP 响应后继续运行。响应中包含锚定该请求的 Job 对象及其唯一标识符。



您应该参考ONTAP Select Deploy 在线文档页面来确定哪些 API 调用是异步操作的。

查询与 API 请求关联的 Job 对象

HTTP 响应中返回的 Job 对象包含多个属性。您可以查询 state 属性来确定请求是否已成功完成。Job对象可以处于以下状态之一：

- 已排队
- 正在运行
- 成功
- 失败

轮询 Job 对象来检测任务的最终状态（成功或失败）时，可以使用两种技术：

- 标准轮询请求立即返回当前作业状态
- 长轮询请求仅当发生以下情况之一时才返回作业状态：
 - 状态更改的时间比轮询请求中提供的日期时间值更近
 - 超时值已过期（1 至 120 秒）

标准轮询和长轮询使用相同的 API 调用来查询作业对象。不过，长轮询请求包含两个查询参数：`poll_timeout`和`last_modified`。`



您应该始终使用长轮询来减少 Deploy 虚拟机上的工作负载。

发出异步请求的一般过程

您可以使用以下高级过程来完成异步 API 调用：

1. 发出异步 API 调用。
2. 收到 HTTP 响应 202，表示成功接受请求。
3. 从响应主体中提取 Job 对象的标识符。
4. 在循环内，每次循环执行以下操作：
 - a. 使用长轮询请求获取作业的当前状态
 - b. 如果作业处于非终止状态（排队、运行），则再次执行循环。
5. 当作业达到最终状态（成功、失败）时停止。

使用浏览器访问

在使用浏览器访问ONTAP Select Deploy API 之前

在使用部署在线文档页面之前，您应该注意几件事。

部署计划

如果您打算在执行特定部署或管理任务的过程中发出 API 调用，则应考虑制定部署计划。这些计划可以是正式的，也可以是非正式的，通常包含您的目标和要使用的 API 调用。有关更多信息，请参阅使用部署 REST API 的工作流程。

JSON 示例和参数定义

文档页面上的每个 API 调用都采用一致的格式进行描述。内容包括实现说明、查询参数和 HTTP 状态代码。此外，您还可以显示 API 请求和响应中使用的 JSON 的详细信息，如下所示：

- 示例值 如果您在 API 调用中点击“示例值”，则会显示该调用的典型 JSON 结构。您可以根据需要修改该示例，并将其用作请求的输入。
- 模型如果单击“模型”，则会显示 JSON 参数的完整列表，其中包含每个参数的描述。

发出 API 调用时需谨慎

您使用部署文档页面执行的所有 API 操作均为实时操作。请务必小心，切勿误创建、更新或删除配置或其他数据。

访问ONTAP Select Deploy 文档页面

您必须访问ONTAP Select Deploy 在线文档页面才能显示 API 文档，以及手动发出 API 调用。

开始之前

您必须具备以下条件：

- ONTAP Select Deploy 虚拟机的 IP 地址或域名
- 管理员的用户名和密码

步骤

1. 在浏览器中输入 URL，然后按 **Enter**：

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

2. 使用管理员用户名和密码Sign in。

结果

部署文档网页显示按类别组织的调用，位于页面底部。

了解并执行ONTAP Select Deploy API 调用

所有 API 调用的详细信息均以通用格式记录并显示在ONTAP Select Deploy 在线文档网页上。通过了解单个 API 调用，您可以访问和解读所有 API 调用的详细信息。

开始之前

您必须登录到ONTAP Select Deploy 在线文档网页。您必须拥有在创建ONTAP Select集群时为其分配的唯一标识符。

关于此任务

您可以使用ONTAP Select集群的唯一标识符检索描述该集群的配置信息。在本例中，所有被归类为“廉价”的字段都会返回。但是，最佳做法是，您应该只请求所需的特定字段。

步骤

1. 在主页上，滚动到底部并单击*Cluster*。
2. 单击 **GET /clusters/{cluster_id}** 可显示用于返回有关ONTAP Select集群信息的 API 调用的详细信息。

工作流程

使用ONTAP Select Deploy API 工作流之前

您应该准备好审查和使用工作流程。

了解工作流程中使用的 API 调用

ONTAP Select在线文档页面包含每个 REST API 调用的详细信息。工作流示例中使用的每个 API 调用仅包含您在文档页面上找到该调用所需的信息，因此无需在此重复这些详细信息。找到特定的 API 调用后，您可以查看该调用的完整详细信息，包括输入参数、输出格式、HTTP 状态代码和请求处理类型。

工作流程中的每个 API 调用都包含以下信息，以帮助在文档页面上找到该调用：

- 类别：API 调用在文档页面上按功能相关区域或类别进行组织。要查找特定的 API 调用，请滚动到页面底部，然后点击相应的 API 类别。
- HTTP 动词 HTTP 动词标识对资源执行的操作。每个 API 调用都通过单个 HTTP 动词执行。
- 路径 路径决定了在执行调用时，操作所针对的具体资源。路径字符串会附加到核心 URL 中，以形成标识资源的完整 URL。

构建 URL 以直接访问 REST API

除了ONTAP Select文档页面外，您还可以通过 Python 等编程语言直接访问 Deploy REST API。在这种情况下，核心 URL 与访问在线文档页面时使用的 URL 略有不同。直接访问 API 时，必须在域和端口字符串后附加 /api。例如：

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

工作流程 1：在 ESXi 上创建ONTAP Select单节点评估集群

您可以在由 vCenter 管理的 VMware ESXi 主机上部署单节点ONTAP Select集群。该集群是使用评估许可证创建的。

集群创建工作流程在以下情况下有所不同：

- ESXi 主机不受 vCenter 管理（独立主机）
- 集群内使用多个节点或主机
- 集群已部署在生产环境中，并已购买许可证
- 使用 KVM 虚拟机管理程序代替 VMware ESXi

1.注册 vCenter 服务器凭据

部署到由 vCenter 服务器管理的 ESXi 主机时，必须在注册主机之前添加凭据。然后，Deploy 管理实用程序可以使用该凭据向 vCenter 进行身份验证。

类别	HTTP 动词	路径
部署	POST	/安全/凭证

卷曲

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

JSON 输入 (步骤 1)

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

加工类型

异步

输出

- 位置响应标头中的凭证 ID
- 作业对象

2.注册虚拟机管理程序主机

您必须添加一个虚拟机管理程序主机，其中包含ONTAP Select节点的虚拟机将在该主机上运行。

类别	HTTP 动词	路径
集群	POST	/主机

卷曲

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

JSON 输入 (步骤 2)

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

加工类型

异步

输出

- 位置响应标头中的主机 ID
- 作业对象

3.创建集群

创建ONTAP Select集群时，将注册基本集群配置并由 Deploy 自动生成节点名称。

类别	HTTP 动词	路径
集群	POST	/集群

卷曲

对于单节点集群，查询参数 node_count 应设置为 1。

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

JSON 输入 (步骤 3)

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

加工类型

同步

输出

- 位置响应标头中的集群 ID

4.配置集群

在配置集群时，您必须提供几个属性。

类别	HTTP 动词	路径
集群	修补	/集群/{cluster_id}

卷曲

您必须提供集群 ID。

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

JSON 输入（步骤 4）

```
{
  "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"}
}
```

加工类型

同步

输出

无

5.检索节点名称

创建集群时，Deploy 管理实用程序会自动生成节点标识符和名称。您必须先检索分配的 ID，然后才能配置节点。

类别	HTTP 动词	路径
集群	GET	/集群/{cluster_id}/节点

卷曲

您必须提供集群 ID。

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

加工类型

同步

输出

- 数组记录每个描述具有唯一 ID 和名称的单个节点

6.配置节点

您必须提供节点的基本配置，这是用于配置节点的三个 API 调用中的第一个。

类别	HTTP 动词	路径
集群	路径	/集群/{cluster_id}/节点/{node_id}

卷曲

您必须提供集群 ID 和节点 ID。

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step06 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID'
```

JSON 输入（步骤 6）

您必须提供ONTAP Select节点将运行的主机 ID。

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

加工类型

同步

输出

无

7.检索节点网络

您必须识别单节点集群中节点使用的数据和管理网络。单节点集群不使用内部网络。

类别	HTTP 动词	路径
集群	GET	/集群/{cluster_id}/节点/{node_id}/网络

卷曲

您必须提供集群 ID 和节点 ID。

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

加工类型

同步

输出

- 两条记录的数组，每条记录描述节点的单个网络，包括唯一 ID 和用途

8.配置节点网络

您必须配置数据和管理网络。单节点集群不使用内部网络。



发出以下 API 调用两次，每个网络一次。

类别	HTTP 动词	路径
集群	修补	/集群/{cluster_id}/节点/{node_id}/网络/{network_id}

卷曲

您必须提供集群 ID、节点 ID 和网络 ID。

```
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'
```

JSON 输入 (步骤 8)

您需要提供网络的名称。

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

加工类型

同步

输出
无

9.配置节点存储池

配置节点的最后一步是连接存储池。您可以通过 vSphere Web Client 或 Deploy REST API 确定可用的存储池。

类别	HTTP 动词	路径
集群	修补	/集群/{cluster_id}/节点/{node_id}/网络/{network_id}

卷曲
您必须提供集群 ID、节点 ID 和网络 ID。

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

JSON 输入（步骤 9）

池容量为 2 TB。

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

加工类型
同步

输出
无

10.部署集群

集群和节点配置完成后，就可以部署集群了。

类别	HTTP 动词	路径
集群	POST	/集群/{cluster_id}/部署

卷曲
您必须提供集群 ID。

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

JSON 输入 (步骤 10)

您必须提供ONTAP管理员帐户的密码。

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

加工类型

异步

输出

- 作业对象

相关信息

["部署ONTAP Select集群的 90 天评估实例"](#)

使用 Python 访问

在使用 **Python** 访问ONTAP Select Deploy API 之前

在运行示例 Python 脚本之前，您必须准备环境。

在运行 Python 脚本之前，必须确保环境配置正确：

- 必须安装 Python2 的最新适用版本。示例代码已使用 Python2 进行了测试。它们应该也可以移植到 Python3，但尚未进行兼容性测试。
- 必须安装 Requests 和 urllib3 库。您可以根据自己的环境使用 pip 或其他 Python 管理工具。
- 运行脚本的客户端工作站必须具有对ONTAP Select Deploy 虚拟机的网络访问权限。

此外，您还必须具备以下信息：

- Deploy 虚拟机的 IP 地址
- Deploy 管理员帐户的用户名和密码

了解ONTAP Select Deploy 的 Python 脚本

示例 Python 脚本可帮助您执行多种不同的任务。在实际部署 Deploy 实例之前，您应该先了解这些脚本。

常见的设计特征

这些脚本的设计具有以下共同特点：

- 从客户端计算机上的命令行界面执行 您可以从任何正确配置的客户端计算机上运行 Python 脚本。有关更多信息，请参阅“开始之前”。
- 接受 CLI 输入参数 每个脚本都通过输入参数在 CLI 进行控制。
- 读取输入文件 每个脚本都会根据其用途读取一个输入文件。创建或删除集群时，必须提供 JSON 配置文件。添加节点许可证时，必须提供有效的许可证文件。
- 使用通用支持模块 通用支持模块 *deploy_requests.py* 包含一个类。它被每个脚本导入和使用。

创建集群

您可以使用脚本 *cluster.py* 创建 ONTAP Select 集群。根据 CLI 参数和 JSON 输入文件的内容，您可以根据部署环境修改该脚本，如下所示：

- 虚拟机管理程序：您可以部署到 ESXi 或 KVM（取决于 Deploy 版本）。部署到 ESXi 时，虚拟机管理程序可以由 vCenter 管理，也可以作为独立主机。
- 集群大小 您可以部署单节点或多节点集群。
- 评估或生产许可证 您可以使用评估或购买的生产许可证来部署集群。

该脚本的 CLI 输入参数包括：

- Deploy 服务器的主机名或 IP 地址
- 管理员用户帐户的密码
- JSON 配置文件的名称
- 消息输出的详细标志

添加节点许可证

如果您选择部署生产集群，则必须使用脚本“*add_license.py*”为每个节点添加许可证。您可以在部署集群之前或之后添加许可证。

该脚本的 CLI 输入参数包括：

- Deploy 服务器的主机名或 IP 地址
- 管理员用户帐户的密码
- 许可证文件的名称
- 具有添加许可证权限的 ONTAP 用户名
- ONTAP 用户的密码

删除集群

您可以使用脚本 *delete_cluster.py* 删除现有的 ONTAP Select 集群。

该脚本的 CLI 输入参数包括：

- Deploy 服务器的主机名或 IP 地址
- 管理员用户帐户的密码
- JSON 配置文件的名称

Python 代码示例

用于创建ONTAP Select集群的脚本

您可以使用以下脚本根据脚本中定义的参数和 JSON 输入文件创建集群。

```
#!/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)
```

用于创建ONTAP Select集群的脚本的 JSON

使用 Python 代码示例创建或删除ONTAP Select集群时，必须提供一个 JSON 文件作为脚本的输入。您可以根据部署计划复制和修改相应的 JSON 示例。

ESXi 上的单节点集群

```
{  
  "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
      }
    ]
  }
}
]
}

```

使用 vCenter 在 ESXi 上进行单节点集群

```

{
  "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
        }
      ]
    }
  }
}

```

KVM 上的单节点集群

```

{
  "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
    }
  ]
}
]
}

```

用于添加ONTAP Select节点许可证的脚本

您可以使用以下脚本为ONTAP Select节点添加许可证。

```

#!/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)

```

用于删除ONTAP Select集群的脚本

您可以使用以下 CLI 脚本删除现有集群。

```

#!/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/{}'.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)

```

ONTAP Select的通用支持 Python 模块

所有 Python 脚本都使用单个模块中的通用 Python 类。

```

#!/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}

```

用于调整ONTAP Select集群节点大小的脚本

您可以使用以下脚本调整ONTAP Select集群中节点的大小。

```

#!/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())

```

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 <http://www.netapp.com/TM> 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。