利用空闲实现自动化
ONTAP Select

NetApp
April 01, 2025

目录

利用空闲实现自动化 1
概念 1
REST Web 服务基础 1
如何访问 Deploy API 2
部署 API 版本控制 2
基本操作特征 2
请求和响应 API 事务 4
使用作业对象进行异步处理 7
使用浏览器访问 8
在使用浏览器访问 API 之前 8
访问Deploy文档页面 8
了解并执行API调用 9
工作流进程 9
在使用 API 工作流之前 9
工作流1：在ESXi上创建单节点评估集群 10
使用 Python 访问 17
在使用 Python 访问 API 之前 17
了解Python脚本 17
Python 代码示例 19
用于创建集群的脚本 19
用于创建集群的脚本的 JSON 26
用于添加节点许可证的脚本 30
用于删除集群的脚本 34
通用支持模块 36
用于调整集群节点大小的脚本 40

利用空闲实现自动化

概念

REST Web 服务基础

表述性状态传输（ Representational State Transfer ， REST ）是一种用于创建分布式 Web 应用程序的模式。在设计 Web 服务 API 时，它会建立一组技术和最佳实践，用于公开基于服务器的资源并管理其状态。它使用主流协议和标准为部署和管理 ONTAP Select 集群提供了灵活的基础。

架构和传统限制

而其余内容则由 Roy Fielding 博士正式阐述 "Dissertation" 2000 年在 UC Irvine 大会上。它通过一组限制来定义架构模式，这些限制共同改进了基于 Web 的应用程序和底层协议。这些限制会根据使用无状态通信协议的客户端 / 服务器架构建立 RESTful Web 服务应用程序。

资源和状态表示

资源是基于 Web 的系统的基本组件。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。在设计基于 REST Web 服务的应用程序时，首先要完成的任务之一是识别资源。
- 资源状态和关联状态操作的定义
  资源始终处于数量有限的状态之一。必须明确定义状态以及用于影响状态更改的关联操作。

客户端和服务器之间会交换消息，以便根据通用 CRUD （创建，读取，更新和删除）模式访问和更改资源的状态。

URI 端点

必须使用定义明确的寻址方案定义和提供每个 REST 资源。资源所在的端点和标识的端点使用统一资源标识符（ Uniform Resource Identifier ， 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。统一资源定位器（ Uniform Resource Locator ， URL ）是一种用于 Web 服务的 URI 类型，用于标识和访问资源。资源通常以类似于文件目录的分层结构公开。

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有关资源的请求和响应消息的协议。在设计 Web 服务应用程序时， HTTP 动词（例如 GET 和 POST ）会映射到资源以及相应的状态管理操作。

HTTP 为无状态。因此，要在一个事务下关联一组相关请求和响应，追加信息 必须包含在请求 / 响应数据流附带的 HTTP 标头中。

JSON 格式化

虽然信息可以通过多种方式在客户端和服务器之间进行结构化和传输，但最常用的选项（以及 Deploy REST API 中使用的选项）是 JavaScript 对象表示法（ JSON ）。JSON 是一种行业标准，用于以纯文本形式表示简

单数据结构，并用于传输描述资源的状态信息。

如何访问 **Deploy API**

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

Deploy 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。浏览器调用 API 并根据用户界面的设计重新格式化数据。您还可以通过 Deploy 实用程序命令行界面访问此 API 。

ONTAP Select Deploy 联机文档页面

使用浏览器时， ONTAP Select Deploy 联机文档页面提供了一个备用访问点。除了提供直接执行单个 API 调用的方法之外，此页面还包括 API 的详细问题描述 ，包括每个调用的输入参数和其他选项。API 调用分为多个不同的功能区域或类别。

自定义程序

您可以使用多种不同的编程语言和工具访问 Deploy API 。常见选项包括 Python ， Java 和 CURL 。使用 API 的程序，脚本或工具充当 REST Web 服务客户端。通过使用编程语言，您可以更好地了解 API ，并有机会自动执行 ONTAP Select 部署。

部署 **API** 版本控制

ONTAP Select Deploy 附带的 REST API 分配有一个版本号。API 版本号与 Deploy 版本号无关。您应了解您的 Deploy 版本附带的 API 版本，以及此版本可能会对您使用此 API 产生何种影响。

当前版本的 Deploy 管理实用程序包括 REST API 版本 3 。Deploy 实用程序的以往版本包括以下 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 及所有早期版本均包含 REST API 版本 2 。

REST API 版本 2 和 3 不兼容。如果从包含 API 版本 2 的早期版本升级到 Deploy 2.8 或更高版本，则必须更新直接访问此 API 的任何现有代码以及使用命令行界面的任何脚本。

基本操作特征

虽然 REST 建立了一组通用的技术和最佳实践，但每个 API 的详细信息可能因设计选择而异。在使用 ONTAP Select Deploy API 之前，您应了解 API 的详细信息和操作特征。

虚拟机管理程序主机与 **ONTAP Select** 节点

虚拟机管理程序 host_ 是托管 ONTAP Select 虚拟机的核心硬件平台。在虚拟机管理程序主机上部署 ONTAP Select 虚拟机并使其处于活动 ONTAP Select 状态时，该虚拟机将被视为 _vp 节点 _ 。在 Deploy REST API 版本 3 中，主机和节点对象是独立的。这样就可以建立一对多关系，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

创建每个资源实例或对象时，系统会为其分配一个唯一标识符。这些标识符在 ONTAP Select Deploy 的特定实例中具有全局唯一性。发出创建新对象实例的API调用后、关联的ID值将返回到中的调用方 `location` HTTP响应的标题。在引用资源实例时，您可以提取此标识符并在后续调用中使用它。

对象标识符的内容和内部结构可以随时更改。仅当引用关联对象时，才应根据需要在适用的 API 调用上使用标识符。

请求标识符

每个成功的 API 请求都会分配一个唯一标识符。此标识符将在中返回 `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 请求的主要方式有两种：

- 同步
  服务器立即执行请求、并使用状态代码200、201或204进行响应。
- 异步
  服务器接受请求并使用状态代码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来完成此请求。最终成功或失败不会立即出现，必须通过其他 API 调用来确定。

确认已完成长时间运行的作业

通常、任何可能需要很长时间才能完成的操作都会使用异步处理
服务器上的后台任务。通过Deploy REST API、每个后台任务都由锚定
作业对象、用于跟踪任务并提供当前状态等信息。作业对象，
创建后台任务后、HTTP响应将返回其唯一标识符。

您可以直接查询作业对象以确定关联 API 调用的成功或失败。
有关追加信息 ，请参见 *asynchronous processing using the Job objection* 。

除了使用作业对象之外、还可以通过其他方法确定的成功或失败
请求，包括：

- 事件消息
  您可以使用随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
  有几个资源保持状态或状态值、您可以查询这些状态或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
  通过网络在Deploy服务器和客户端之间发送的所有流量都会通过TLS进行加密。不支持在未加密的通道上使用 HTTP 协议。支持 TLS 1.2 版。
- HTTP身份验证
  基本身份验证用于每个API事务。每个请求都会添加一个 HTTP 标头，其中包含 base64 字符串中的用户名和密码。

请求和响应 **API** 事务

每次 Deploy API 调用都会作为 HTTP 请求执行给 Deploy 虚拟机，此请求会向客户端生成关联的响应。此请求 / 响应对被视为 API 事务。在使用 Deploy API 之前，您应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

您可以控制如何通过 HTTP 请求中设置的参数处理 API 调用。

请求标题

您必须在 HTTP 请求中包含多个标头，包括：

- 内容类型
  如果请求正文包括JSON、则必须将此标头设置为application/json。
- 接受
  如果响应正文将包括JSON、则必须将此标题设置为application/json。
- 授权
  必须使用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

请求正文的内容因具体调用而异。HTTP 请求正文包含以下内容之一：

- 包含输入变量（例如新集群的名称）的 JSON 对象
- 空

筛选对象

发出使用 GET 的 API 调用时，您可以根据任何属性限制或筛选返回的对象。例如，您可以指定一个要匹配的精确值：

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

除了精确匹配之外，还有其他运算符可用于返回一组值范围内的对象。ONTAP Select 支持如下所示的筛选运算符。

运算符	Description
=	等于
<	小于
>	大于
&lt； =	小于或等于
&gt； =	大于或等于
	或
!	不等于
*	贪婪的通配符

您还可以在查询中使用 null 关键字或其否定（！ null ）来根据是否设置了特定字段返回一组对象。

选择对象字段

默认情况下，使用 GET 发出 API 调用时，仅返回唯一标识一个或多个对象的属性。这组最小的字段可用作每个对象的密钥，并因对象类型而异。您可以通过以下方式使用 fields query 参数选择其他对象属性：

- 低成本字段
  指定 `fields=*` 检索本地服务器内存中维护的对象字段或只需少量处理即可访问的对象字段。
- 昂贵的字段
  指定 `fields=**` 检索所有对象字段、包括需要额外服务器处理才能访问的对象字段。
- 自定义字段选择
  使用 ... `fields=FIELDNAME` 以指定所需的确切字段。请求多个字段时，必须使用逗号分隔值，不能包含空格。

作为最佳实践，您应始终确定所需的特定字段。您只能在需要时检索一组廉价或昂贵的字段。成本低且昂贵的分类由 NetApp 根据内部性能分析确定。给定字段的分类可以随时更改。

对输出集中的对象进行排序

资源收集中的记录将按对象定义的默认顺序返回。您可以使用order_by查询参数以及字段名称和排序方向更改顺序、如下所示：

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

例如，您可以按降序对类型字段排序，然后按升序对 ID 排序：

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

如果包含多个参数，则必须使用逗号分隔各个字段。

分页

使用 GET 发出 API 调用以访问同一类型的对象集合时，默认情况下会返回所有匹配的对象。如果需要，您可以在请求中使用 max_records 查询参数限制返回的记录数。例如：

```
max_records=20
```

如果需要，您可以将此参数与其他查询参数结合使用，以缩小结果集的范围。例如、以下命令最多返回在指定时

间之后生成的10个系统事件：
`time⇒ 2019-04-04T15:41:29.140265Z&max_records=10`

您可以通过问题描述 发送多个请求来分页查看事件（或任何对象类型）。后续的每个 API 调用应根据最后一个结果集中的最新事件使用一个新的时间值。

解释 API 响应

每个 API 请求都会生成对客户端的响应。您可以检查响应以确定是否成功并根据需要检索其他数据。

HTTP 状态代码

下面介绍了 Deploy REST API 使用的 HTTP 状态代码。

代码	含义	Description
200	确定	表示未创建新对象的调用成功。
201	已创建	已成功创建对象；位置响应标头包含对象的唯一标识符。
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	请求错误	此请求输入无法识别或不适当。
403.	已禁止	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.	不允许使用此方法	此资源不支持请求中的 HTTP 动词。
409.	冲突	尝试创建对象失败，因为此对象已存在。
500	内部错误	服务器发生一般内部错误。
501.	未实施	此 URI 已知，但无法执行此请求。

响应标头

Deploy 服务器生成的 HTTP 响应包含多个标头，其中包括：

- 请求ID
  每个成功的API请求都会分配一个唯一的请求标识符。
- location
  创建对象时、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 请求关联的响应内容因对象，处理类型以及请求的成功或失败而异。响应正文将在 JSON 中呈现。

- 单个对象
  可以根据请求返回一个对象并显示一组字段。例如，您可以使用 GET 使用唯一标识符检索集群的选定属性。
- 多个对象
  可以从一个资源收集返回多个对象。在所有情况下、都会使用一致的格式 `num_records` 指示包含对象实例数组的记录和记录的数量。例如，您可以检索特定集群中定义的所有节点。

- 作业对象
  如果异步处理 API 调用，则会返回作业对象，用于将后台任务固定。例如，用于部署集群的 POST 请求会异步处理并返回作业对象。
- 错误对象
  如果发生错误，则始终返回 Error 对象。例如，在尝试创建名称已存在的集群时，您将收到错误消息。
- 空
  在某些情况下、不会返回任何数据、并且响应正文为空。例如，使用 delete 删除现有主机后，响应正文为空。

使用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。

使用作业对象描述的异步请求

发出异步运行的 API 调用后， HTTP 响应代码 202 表示此请求已成功验证并被接受，但尚未完成。此请求将作为后台任务进行处理，在对客户端进行初始 HTTP 响应后，此任务将继续运行。响应包括作业对象锁定请求，包括其唯一标识符。

您应参阅 ONTAP Select Deploy 联机文档页面以确定哪些 API 调用异步运行。

查询与**API**请求关联的作业对象

HTTP 响应中返回的作业对象包含多个属性。您可以查询 state 属性以确定请求是否成功完成。作业对象可以处于以下状态之一：

- 已排队
- 正在运行
- success
- 失败

在轮询作业对象以检测任务的终端状态时，可以使用两种方法：成功或失败：

- 标准轮询请求
  当前作业状态将立即返回
- 长时间轮询请求
  只有在发生以下情况之一时、才会返回作业状态：
  - 状态的更改日期比轮询请求提供的日期时间值更晚
  - 超时值已过期（ 1 到 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。但是、长轮询请求包含两个查询参数：`poll_timeout` 和 `last_modified`。

您应始终使用长轮询来减少 Deploy 虚拟机上的工作负载。

用于发出异步请求的常规操作步骤

您可以使用以下高级操作步骤完成异步 API 调用。

1. 问题描述异步 API 调用。
2. 接收表示已成功接受请求的 HTTP 响应 202 。
3. 从响应正文中提取作业对象的标识符。
4. 在环路中，在每个周期中执行以下操作：
   a. 获取具有长时间轮询请求的作业的当前状态
   b. 如果作业处于非终端状态（已排队，正在运行），请重新执行环路。
5. 当作业达到终端状态（成功，失败）时停止。

使用浏览器访问

在使用浏览器访问 **API** 之前

在使用 Deploy 联机文档页面之前，您应注意以下几点。

部署计划

如果要在执行特定部署或管理任务时调用问题描述 API ，则应考虑创建部署计划。这些计划可以是正式的或非正式的，通常包含您的目标和要使用的 API 调用。有关详细信息，请参见使用 Deploy REST API 的工作流流程。

JSON 示例和参数定义

每个 API 调用都会在文档页面上使用一致的格式进行说明。其中包括实施说明，查询参数和 HTTP 状态代码。此外，您还可以显示 API 请求和响应所使用的 JSON 的详细信息，如下所示：

- 示例值
  如果在API调用上单击_expl示范 值_、则会显示此调用的典型JSON结构。您可以根据需要修改此示例并将其用作请求的输入。
- 型号
  如果单击_Model_、则会显示JSON参数的完整列表、其中每个参数都有一个问题描述。

发出 **API** 调用时的注意事项

使用 Deploy 文档页面执行的所有 API 操作均为实时操作。请注意，不要错误地创建，更新或删除配置或其他数据。

访问**Deploy**文档页面

您必须访问 ONTAP Select Deploy 联机文档页面才能显示 API 文档，并手动对 API 调用执行问题描述 。

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. 在浏览器中键入 URL 并按 * 输入 * ：

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

2. 使用管理员用户名和密码登录。

结果

此时将显示 Deploy 文档网页，页面底部将按类别组织调用。

了解并执行**API**调用

所有 API 调用的详细信息均采用通用格式记录并显示在 ONTAP Select Deploy 联机文档网页上。通过了解单个 API 调用，您可以访问和解释所有 API 调用的详细信息。

开始之前

您必须登录到 ONTAP Select Deploy 联机文档网页。您必须具有创建集群时为 ONTAP Select 集群分配的唯一标识符。

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。在此示例中，将返回归类为 " 低成本 " 的所有字段。但是，作为最佳实践，您应仅请求所需的特定字段。

步骤

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

工作流进程

在使用 **API** 工作流之前

您应准备好查看和使用工作流流程。

了解工作流中使用的**API**调用

ONTAP Select 联机文档页面包含每个 REST API 调用的详细信息。工作流示例中使用的每个 API 调用都仅包含在文档页面上查找此调用所需的信息，而不是在此处重复这些详细信息。找到特定 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。

工作流中的每个 API 调用都包含以下信息，以帮助您在文档页面上查找此调用：

- 类别
  API 调用会在文档页面上按功能相关的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，然后单击相应的 API 类别。

- HTTP动词
  HTTP 动词用于标识对资源执行的操作。每个 API 调用都通过一个 HTTP 动词来执行。
- 路径
  路径用于确定在执行调用时操作适用场景所使用的特定资源。路径字符串会附加到核心 URL 中，以形成用于标识资源的完整 URL 。

构建一个**URL**以直接访问**REST API**

除了 ONTAP Select 文档页面之外，您还可以直接通过 Python 等编程语言访问 Deploy REST API 。在这种情况下，核心 URL 与访问联机文档页面时使用的 URL 稍有不同。直接访问 API 时，必须将 /API 附加到域和端口字符串。例如：

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

工作流1：在ESXi上创建单节点评估集群

您可以在 vCenter 管理的 VMware ESXi 主机上部署单节点 ONTAP Select 集群。此时将使用评估版许可证创建集群。

集群创建工作流在以下情况下有所不同：

- ESXi 主机不受 vCenter （独立主机）管理
- 集群中使用多个节点或主机
- 集群使用已购买的许可证部署在生产环境中
- 使用的是KVM虚拟机管理程序、而不是VMware ESXi

◦ 从ONTAP Select 9.10.1开始、您无法再在KVM虚拟机管理程序上部署新集群。

◦ 从ONTAP Select 9.11.1开始、除了脱机和删除功能之外、现有KVM集群和主机不再具有所有易管理性功能。

1. 注册 **vCenter Server** 凭据

在部署到由 vCenter Server 管理的 ESXi 主机时，您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凭据向 vCenter 进行身份验证。

类别	**HTTP**动词	路径
部署	发布	/security/credentials

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 **01** ）

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

处理类型

异步

输出

- 位置响应标头中的凭据 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 节点的虚拟机将在其中运行。

类别	**HTTP**动词	路径
集群	发布	/hosts

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（第 **02** 步）

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

处理类型

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，系统会注册基本集群配置，并且 Deploy 会自动生成节点名称。

类别	**HTTP**动词	路径
集群	发布	/clusters

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（第 **03** 步）

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

处理类型

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，必须提供多个属性。

类别	**HTTP**动词	路径
集群	patch	/clusters/ ｛ 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（第 **04** 步）

```
{
  "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"}
}
```

处理类型

同步

输出

无

5. 检索节点名称

Deploy 管理实用程序会在创建集群时自动生成节点标识符和名称。在配置节点之前，必须检索分配的 ID 。

类别	**HTTP**动词	路径
集群	获取	/clusters/ ｛ 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'
```

处理类型

同步

输出

- 每个阵列都会记录一个节点，该节点使用唯一的 ID 和名称

6. 配置节点

您必须为节点提供基本配置，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	/clusters/ ｛ cluster_id ｝ /nodes/ ｛ 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（第 06 步）

您必须提供要运行 ONTAP Select 节点的主机 ID 。

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

处理类型

同步

输出

无

7. 检索节点网络

您必须确定单节点集群中的节点使用的数据和管理网络。内部网络不用于单节点集群。

类别	**HTTP**动词	路径
集群	获取	/clusters/ ｛cluster_id｝ /nodes/ ｛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'
```

处理类型

同步

输出

- 由两个记录组成的数组，每个记录分别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。内部网络不用于单节点集群。

对以下 API 调用执行问题描述 两次，每个网络一次。

类别	**HTTP**动词	路径
集群	patch	/clusters/ ｛cluster_id｝ /nodes/ ｛node_id｝ /networks/ ｛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（第 **08** 步）

您需要提供网络名称。

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

处理类型

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 （可选）确定可用存储池。

类别	**HTTP**动词	路径
集群	patch	/clusters/ ｛cluster_id｝ /nodes/ ｛node_id｝ /networks/ ｛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（第 09 步）

池容量为 2 TB 。

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

处理类型

同步

输出

无

部署集群

配置集群和节点后，您可以部署集群。

类别	HTTP动词	路径
集群	发布	/clusters/ ｛cluster_id｝ /Deploy

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（第 10 步）

您必须提供 ONTAP 管理员帐户的密码。

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

处理类型

异步

输出

- 作业对象

使用 Python 访问

在使用 Python 访问 API 之前

在运行示例 Python 脚本之前，您必须准备好环境。

在运行 Python 脚本之前，您必须确保已正确配置环境：

- 必须安装最新适用版本的 Python2 。
  这些示例代码已使用 Python2 进行了测试。它们还应可移植到 Python3 ，但尚未经过兼容性测试。
- 必须安装请求和 urllib3 库。
  您可以根据环境需要使用 pip 或其他 Python 管理工具。
- 运行脚本的客户端工作站必须能够通过网络访问 ONTAP Select Deploy 虚拟机。

此外，您还必须具有以下信息：

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

了解Python脚本

您可以使用示例 Python 脚本执行多种不同的任务。在实时 Deploy 实例中使用这些脚本之前，您应先了解这些脚本。

通用设计特征

这些脚本的设计具有以下常见特征：

- 在客户端计算机上从命令行界面执行
  您可以从任何已正确配置的客户端计算机运行Python脚本。有关详细信息，请参见 _ 开始之前 _ 。
- 接受CLI输入参数
  每个脚本都通过CLI中的输入参数进行控制。
- 读取输入文件
  每个脚本都会根据其用途读取输入文件。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。它会导入并由每个脚本使用。

创建集群

您可以使用 cluster.py 脚本创建 ONTAP Select 集群。根据 CLI 参数和 JSON 输入文件的内容，您可以按如下所示将脚本修改到部署环境：

- 从ONTAP Select 9.10.1开始、您无法再在KVM虚拟机管理程序上部署新集群。
- 从ONTAP Select 9.11.1开始、除了脱机和删除功能之外、现有KVM集群和主机不再具有所有易管理性功能。

- 虚拟机管理程序
  您可以将部署到ESXi或KVM (具体取决于Deploy版本)。部署到 ESXi 时，虚拟机管理程序可以由 vCenter 进行管理，也可以是独立主机。
- 集群大小
  您可以部署单节点或多节点集群。
- 评估版或生产版许可证
  您可以使用评估版或已购买许可证部署集群以用于生产环境。

此脚本的 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* 为每个节点添加一个许可证。您可以在部署集群之前或之后添加许可证。

此脚本的 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 ONTAP Select 集群。

此脚本的 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

用于创建集群的脚本

您可以使用以下脚本根据脚本和 JSON 输入文件中定义的参数创建集群。

```
#!/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)
```

用于创建集群的脚本的 JSON

在使用 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** 的 **ESXi** 上的单节点集群

```
{
  "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

- 从ONTAP Select 9.10.1开始、您无法再在KVM虚拟机管理程序上部署新集群。
- 从ONTAP Select 9.11.1开始、除了脱机和删除功能之外、现有KVM集群和主机不再具有所有易管理性功能。

```
{
  "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)
```

用于删除集群的脚本

您可以使用以下命令行界面脚本删除现有集群。

```
#!/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/{}?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)
```

通用支持模块

所有 Python 脚本都在一个模块中使用一个通用 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())
```