通过 **REST** 实现自动化

ONTAP Select

NetApp
May 07, 2026

目录

通过 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

表示状态传输 (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，它建立了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 集群提供灵活的基础。

架构和经典约束

REST 由 Roy Fielding 于 2000 年在 UC Irvine 的博士 "论文"中正式阐述。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 的应用程序和底层协议。这些约束基于使用无状态通信协议的客户端/服务器架构建立了 RESTful Web 服务应用程序。

资源和状态表示

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

- 标识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。设计基于 REST Web 服务的应用程序的首要任务之一是识别资源。
- 资源状态和相关状态操作的定义 资源始终处于有限数量的状态之一。必须明确界定状态以及用于影响状态更改的相关操作。

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

URI 端点

必须使用明确定义的编址方案来定义和提供每个 REST 资源。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的名称。统一资源定位器 (URL) 是一种与 Web 服务一起使用的 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于交换有关资源的请求和响应消息的协议。作为设计 Web 服务应用程序的一部分，HTTP 谓词（如 GET 和 POST）映射到资源和相应的状态管理操作。

HTTP 是无状态的。因此，要在一个事务下关联一组相关的请求和响应，必须在请求/响应数据流所携带的 HTTP 标头中包含其他信息。

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进行结构化和传输，但最受欢迎的选项（以及与 Deploy REST API 一起使用的选项）是 JavaScript 对象表示法 (JSON)。JSON 是以纯文本表示简单数据结构的行业标准，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，ONTAP Select Deploy API 可以通过几种不同的方式访问。

> ONTAP Select Deploy 附带的 REST API 分配了一个版本号。API 版本号与 Deploy 发布版本号无关。ONTAP Select 9.17.1 Deploy 管理实用程序包括 REST API 版本 3。

部署实用程序本机用户界面

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 API。

ONTAP Select Deploy 在线文档页面

使用浏览器时，ONTAP Select Deploy 在线文档页面提供了另一个接入点。除了提供直接执行单个 API 调用的方法外，该页面还包括 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。

自定义程序

您可以使用多种不同的编程语言和工具中的任何一种来访问 Deploy API。热门选择包括 Python、Java 和 cURL。使用 API 的程序、脚本或工具充当 REST Web 服务客户端。使用编程语言可以让您更好地了解 API，并提供自动化 ONTAP Select 部署的机会。

ONTAP Select Deploy API 基本操作特性

虽然 REST 建立了一套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 API 之前，您应该了解 ONTAP Select Deploy API 的详细信息和操作特性。

Hypervisor 主机与 **ONTAP Select** 节点

hypervisor host 是托管 ONTAP Select 虚拟机的核心硬件平台。当 ONTAP Select 虚拟机部署并在 hypervisor host 上处于活动状态时，该虚拟机被视为 *ONTAP Select* 节点。使用 Deploy REST API 版本 3，主机和节点对象是分开且不同的。这允许一对多关系，其中一个或多个 ONTAP Select 节点可以在同一 hypervisor host 上运行。

对象标识符

每个资源实例或对象在创建时都分配有唯一的标识符。这些标识符在 ONTAP Select Deploy 的特定实例中是全局唯一的。在发出创建新对象实例的 API 调用后，关联的 id 值将在 HTTP 响应的 `location` 标头中返回给调用方。您可以提取标识符，并在引用资源实例时将其用于后续调用。

> 对象标识符的内容和内部结构可以随时更改。在引用关联对象时，仅应根据需要在适用的 API 调用中使用标识符。

请求标识符

每个成功的 API 请求都会分配一个唯一标识符。标识符在关联 HTTP 响应的 `request-id`标头中返回。您可以使用请求标识符来共同引用单个特定 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接收的 HTTP 请求有两种主要方式：

- 同步 服务器立即执行请求，并以状态代码 200、201 或 204 响应。
- 异步 服务器接受请求并以状态代码 202 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请求。最终的成功或失败无法立即获得，必须通过其他 API 调用来确定。

确认长时间运行作业的完成

一般来说，任何可能需要很长时间才能完成的操作都会在服务器上使用后台任务进行异步处理。使用 Deploy REST API，每个后台任务都由 Job 对象锚定，该对象跟踪任务并提供信息，例如当前状态。在创建后台任务后，在 HTTP 响应中返回 Job 对象，包括其唯一标识符。

您可以直接查询 Job 对象，以确定关联 API 调用的成功或失败。有关其他信息，请参阅_使用 Job 对象的异步处理_。

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 您可以使用与原始响应一起返回的请求 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，并且在调试错误条件时也很有用。
- 资源状态或状态 一些资源维护一个状态或状态值，您可以查询该值以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 Deploy 服务器和客户端之间通过网络发送的所有流量都通过 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 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，其中包括 base64 字符串中的用户名和密码。

ONTAP Select 的请求和响应 API 事务

每个 Deploy API 调用都作为对 Deploy 虚拟机的 HTTP 请求执行，该虚拟机生成对客户端的关联响应。此请求/响应对被视为 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和响应输出的内容。

控制 API 请求的输入变量

您可以通过在 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 请求中包含多个标头，包括：

- content-type 如果请求正文包含 JSON，则必须将此标头设置为 application/json。

- accept 如果响应正文将包含 JSON，则必须将此标头设置为 application/json。
- authorization 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

请求正文的内容因特定调用而异。HTTP 请求正文由以下内容之一组成：

- 具有输入变量的 JSON 对象（例如，新集群的名称）
- 空

筛选对象

在发布使用 GET 的 API 调用时，可以根据任何属性来限制或筛选返回的对象。例如，您可以指定要匹配的确切值：

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

除了精确匹配之外，还有其他运算符可用于返回一系列值范围内的一组对象。ONTAP Select 支持以下所示的过滤运算符。

运算符	问题描述
=	等于
<	小于
>	大于
⇐	小于或等于
>=	大于或等于
	或
!	不等于
*	贪婪通配符

您还可以使用 null 关键字或其否定（!null）作为查询的一部分，根据是否设置特定字段来返回一组对象。

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对象的属性。此最小字段集充当每个对象的键，并根据对象类型而变化。可以通过以下方式使用 fields 查询参数选择其他对象属性：

- 廉价字段 指定 `fields=*`检索维护在本地服务器内存中或需要很少处理才能访问的对象字段。
- 昂贵字段 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字段。
- 自定义字段选择 使用 `fields=FIELDNAME`指定所需的确切字段。请求多个字段时，必须使用不带空格的逗号分隔值。

作为最佳做法，您应始终确定所需的特定字段。您只应在需要时检索一组便宜或昂贵的字段。便宜和昂贵的分类由 NetApp 根据内部性能分析确定。给定字段的分类可以随时更改。

对输出集中的对象进行排序

资源集合中的记录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，必须使用逗号分隔字段。

分页

使用 GET 发出 API 调用以访问相同类型的对象集合时，默认情况下会返回所有匹配的对象。如果需要，您可以使用带有请求的 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相结合，以缩小结果集。例如，以下内容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
```

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

解释 **API** 响应

每个 API 请求都会生成一个回复给客户端的响应。您可以检查响应以确定它是否成功，并根据需要检索其他数据。

HTTP 状态代码

下面介绍 Deploy REST API 使用的 HTTP 状态代码。

代码	含义	问题描述
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；位置响应标头包括此对象的唯一标识符。
202	已接受	已启动长时间运行的后台作业以执行请求，但操作尚未完成。
400	错误请求	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	此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 谓词。
409	冲突	尝试创建对象失败，因为此对象已存在。
500	内部错误	服务器出现常规内部错误。
501	未实现	URI 已知，但无法执行请求。

响应标头

Deploy 服务器生成的 HTTP 响应中包括几个标头，包括：

- request-id 为每个成功的 API 请求分配唯一的请求标识符。
- 位置 创建对象时，位置标头包括新对象的完整 URL，包括唯一对象标识符。

响应正文

与 API 请求相关联的响应的内容因对象、处理类型以及请求的成功或失败而异。响应正文以 JSON 呈现。

- 单个对象 单个对象可以根据请求返回一组字段。例如，您可以使用 GET 来使用唯一标识符检索集群的选定属性。
- 多个对象 可以从资源集合中返回多个对象。在所有情况下，都使用一致的格式， `num_records` 指示记录数量，records 包含对象实例数组。例如，您可以检索特定集群中定义的所有节点。
- Job 对象如果异步处理 API 调用，则返回一个 Job 对象，该对象锚定后台任务。例如，用于部署集群的 POST 请求是异步处理的，并返回 Job 对象。
- Error 对象如果发生错误，则始终返回 Error 对象。例如，当您尝试使用已存在的名称创建集群时，您将收到一个错误。
- 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 **Job** 对象对 **ONTAP Select** 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 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

在进行异步运行的 API 调用后，HTTP 响应代码 202 表示请求已成功验证和接受，但尚未完成。此请求作为后台任务处理，此后台任务在对客户端的初始 HTTP 响应后继续运行。响应包括锚定请求的 Job 对象，包括其唯一标识符。

您应参阅 ONTAP Select Deploy 在线文档页面，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 Job 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。Job 对象可以处于以下状态之一：

- 已排队
- 正在运行
- 成功
- 失败

在轮询 Job 对象以检测任务的终端状态时，您可以使用两种技术，无论是成功还是失败：

- 标准轮询请求 立即返回当前作业状态
- 仅当出现以下情况之一时，才会返回长轮询请求作业状态：
  - 状态的更改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 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`。

应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完成异步 API 调用：

1. 发出异步 API 调用。
2. 接收 HTTP 响应 202，指示成功接受请求。
3. 从响应正文中提取 Job 对象的标识符。
4. 在循环中，在每个周期中执行以下操作：
   a. 使用长轮询请求获取 Job 的当前状态
   b. 如果作业处于非终端状态（已排队、正在运行），请再次执行循环。
5. 当作业达到终止状态（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 Deploy 在线文档页面之前，您应该了解以下几点。

部署计划

如果您打算在执行特定部署或管理任务时发出 API 调用，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 的工作流程。

JSON 示例和参数定义

每个 API 调用在文档页面上使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，您可以按如下所示显示与 API 请求和响应一起使用的 JSON 的详细信息：

- 示例值 如果单击 API 调用上的_示例值_，则会显示调用的典型 JSON 结构。您可以根据需要修改示例，并将其用作请求的输入。
- 模型 如果单击 模型，将显示 JSON 参数的完整列表，其中包含每个参数的说明。

发出 **API** 调用时的注意事项

使用部署文档页面执行的所有 API 操作都是实时操作。您应注意不要错误地创建、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 在线文档页面以显示 API 文档，并手动发出 API 调用。

开始之前

您必须具有下列各项：

- 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，然后按 **Enter** 键：

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

2. 使用管理员用户名和密码 Sign in。

结果

Deploy 文档网页将显示，页面底部按类别组织调用。

了解并执行 **ONTAP Select Deploy API** 调用

所有 API 调用的详细信息都使用 ONTAP Select Deploy 在线文档网页上的通用格式进行记录和显示。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 在线文档网页。您必须拥有创建集群时分配给您的 ONTAP Select 集群的唯一标识符。

关于此任务

您可以使用 ONTAP Select 集群的唯一标识符检索描述 ONTAP Select 集群的配置信息。在此示例中，返回所有分类为廉价的字段。但是，作为最佳做法，您应该只请求需要的特定字段。

步骤

1. 在主页上，滚动到底部并单击 **Cluster**。
2. 单击 **GET /clusters/{cluster_id}** 以显示用于返回有关 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** 工作流程之前

您应准备好查看和使用工作流程。

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 调用不会在此处重复这些详细信息，而是仅包含您在文档页面上定位调用所需的信息。找到特定 API 调用后，您可以查看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。

工作流程中每个 API 调用都包含以下信息，以帮助在文档页面上查找该调用：

- 类别 API 调用在文档页面上组织成与功能相关的区域或类别。要查找特定的 API 调用，请滚动到页面底部，然后单击适用的 API 类别。
- HTTP 谓词 HTTP 谓词标识对资源执行的操作。每个 API 调用都通过单个 HTTP 谓词执行。
- 路径 路径确定在执行调用时操作应用到的特定资源。路径字符串附加到核心 URL 以形成标识资源的完整 URL。

构造 **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 与访问在线文档页面时使用的 URL 略有不同。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。使用评估许可证创建集群。

在以下情况下，集群创建工作流程不同：

- ESXi 主机不由 vCenter 管理（独立主机）
- 集群内使用多个节点或主机
- 集群使用购买的许可证部署在生产环境中
- 使用 KVM 虚拟机管理程序代替 VMware ESXi

1.注册 **vCenter** 服务器凭据

在部署到由 vCenter 服务器管理的 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** 谓词	路径
部署	POST	/security/credentials

Curl

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

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监控程序主机，其中将运行包含 ONTAP Select 节点的虚拟机。

类别	**HTTP** 谓词	路径
集群	POST	/hosts

Curl

```
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 **02**）

```
{
  "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

在您创建 ONTAP Select 群集时，系统会注册基本群集配置，并由 Deploy 自动生成节点名称。

类别	**HTTP** 谓词	路径
集群	POST	/clusters

Curl

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 **03**）

```
{
  "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，必须提供几个属性。

类别	**HTTP** 谓词	路径
集群	PATCH	/clusters/{cluster_id}

Curl

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 **04**）

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

处理类型

同步

输出

无

5.检索节点名称

Deploy 管理实用程序在创建群集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** 谓词	路径
集群	GET	/clusters/{cluster_id}/nodes

Curl

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

- 数组记录，每个记录描述一个具有唯一 ID 和名称的单个节点

6.配置节点

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** 谓词	路径
集群	路径	/clusters/{cluster_id}/nodes/{node_id}

Curl

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 **06**）

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
}
```

处理类型

同步

输出

无

7.检索节点网络

您必须标识单节点集群中节点使用的数据和管理网络。内部网络不与单节点集群一起使用。

类别	**HTTP** 谓词	路径
集群	GET	/clusters/{cluster_id}/nodes/{node_id}/networks

Curl

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

- 两个记录的数组，每个记录描述节点的单个网络，包括唯一 ID 和目的

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** 谓词	路径
集群	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

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 **08**）

您需要提供网络的名称。

```
{
  "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附加存储池。您可以通过 vSphere Web 客户端或可选地通过 Deploy REST API 来确定可用的存储池。

类别	**HTTP** 谓词	路径
集群	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

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 09）

池容量为 2 TB。

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

处理类型

同步

输出

无

10.部署集群

配置集群和节点后，即可部署集群。

类别	**HTTP** 谓词	路径
集群	POST	/clusters/{cluster_id}/deploy

Curl

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"
  }
}
```

处理类型

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

必须在运行示例 Python 脚本之前准备环境。

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。它们也应该可移植到 Python3 ，但尚未经过兼容性测试。
- 必须安装 Requests 和 urllib3 库。您可以根据您的环境使用 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 ONTAP Select Deploy 的 Python 脚本

示例 Python 脚本允许您执行几种不同的任务。在实时 Deploy 实例中使用脚本之前，应先了解这些脚本。

通用设计特点

脚本的设计具有以下共同特征：

- 从客户端计算机的命令行界面执行 您可以从任何正确配置的客户端计算机运行 Python 脚本。有关详细信息 ，请参见_在开始之前_。
- 接受 CLI 输入参数 每个脚本通过输入参数在 CLI 上进行控制。

- 读取输入文件 每个脚本根据其目的读取输入文件。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。它被导入并由每个脚本使用。

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，您可以将脚本修改为适合您的部署环境，如下所示：

- 虚拟机监控程序 您可以部署到 ESXI 或 KVM（取决于 Deploy 版本）。部署到 ESXi 时，虚拟机监控程序可以由 vCenter 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。
- 评估或生产许可证 您可以部署具有评估或购买的生产许可证的集群。

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* 为每个节点添加许可证。您可以在部署集群之前或之后添加许可证。

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。

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基于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 群集时，必须提供 JSON 文件作为脚本的输入。您可以根据部署计划复制和修改适当的 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
            }
          ]
        }
      }
    ]
}
```

在 **ESXi** 上使用 **vCenter** 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

```
{
  "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
          }
        ]
      }
    }
  ]
}
```

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)
```

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/{}?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 脚本在单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}
```

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 群集中节点的大小。

```
#!/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())
```