Red Hat Ansible Automation Platform 2.4

自动化控制器 API 概述

自动化控制器 API 的开发人员概述

Last Updated: 2024-06-19

Red Hat Ansible Automation Platform 2.4 自动化控制器 API 概述

自动化控制器 API 的开发人员概述

法律通告

摘要

探索自动化控制器 API 概述以了解简化的自动化解决方案，使开发人员和管理员能够有效地管理基础架构。

目录

前言 3

对红帽文档提供反馈 4

第 1 章 API 的可用工具 5

第 2 章 使用 API 浏览 6

第 3 章 API 中的使用惯例 7

第 4 章 API 中的排序 8

第 5 章 使用搜索查询字符串参数 9

第 6 章 在 API 中过滤 10
6.1. API 中的高级查询 10
6.2. 字段查找 11

第 7 章 在 API 中使用分页 13

第 8 章 访问资源 14
8.1. 配置设置 14
8.2. 标识符格式协议 15

第 9 章 只读字段 17

第 10 章 在 API 中进行身份验证 18
10.1. 使用会话身份验证 18
10.2. 基本身份验证（BASIC AUTHENTICATION） 20
10.3. OAUTH 2 令牌身份验证 20
10.4. 单点登录身份验证 23

前言

感谢您对 Red Hat Ansible Automation Platform 的关注。Ansible Automation Platform 通过增加控制、知识、协调基于 Ansible 的环境，帮助团队管理复杂的多层部署。

自动化控制器 API 概述着重帮助您了解自动化控制器 API。

对红帽文档提供反馈

如果您对本文档有任何改进建议，或发现了任何错误，请通过 https://access.redhat.com 联系技术支持，以使用 **docs-product** 组件在 Ansible Automation Platform JIRA 项目中创建一个问题。

第 1 章 API 的可用工具

Representational State Transfer (REST)依赖于无状态、客户端-服务器和可缓存的通信协议，通常是 HTTP 协议。

您可能会发现，了解用户界面调用了哪些 API 会按顺序发出。要做到这一点，您可以使用带有开发人员插件的 Firebug 或 Chrome 中的 UI。

另一个选择是使用 Charles 代理。这提供了一个可视化工具，您可能会发现有帮助。虽然它是商业软件，可以作为操作系统 X 代理插入，并截获 Web 浏览器、curl 和其他 API 用户的请求。

其他选项包括：

- Fiddler
- mitmproxy
- 实时 HTTP 标头 FireFox 扩展
- Paros

第 2 章 使用 API 浏览

REST API 通过 URI 路径访问资源（数据实体）。

流程

1. 在网页浏览器中进入自动化控制器 REST API，请访问以下地址：
   https://<server name>/api/
2. 单击 "**当前版本**"或"**可用版本**"旁边的 "**v2**" 链接。自动化控制器支持 API 的版本 2。
3. 仅使用 **/api/** 端点执行 **GET** 以获取 **current_version**，这是推荐的版本。
4. 点导航菜单中的  图标，了解该特定 API 端点的访问方法以及使用这些方法时返回哪些数据。
5. 通过在各种文本字段中格式化 JSON，在特定 API 页面上使用 **PUT** 和 **POST** 动词。

您还可以在 **/api/v2/settings/changed/** 端点中从工厂默认值查看更改的设置。它反映了您在 API 浏览器中所做的更改，而不是静态设置文件中更改的设置。

第 3 章 API 中的使用惯例

自动化控制器使用标准的 REST API，根于服务器上的 **/api/**。

API 因兼容性原因而版本化。您可以通过查询 **/api/** 来查看可用的 API 版本。

您可能需要指定内容或类型 **POST** 或 **PUT** 请求：

- **PUT** ：更新特定资源（按标识符）或资源集合。如果您事先知道资源标识符，也可以使用 **PUT** 来创建特定资源。
- **POST** ：创建新资源。也充当操作的捕获全部动词，不适合于其他类别。

没有以 "/" 结尾的所有 URI 都收到 301 重定向。

注意

保留附加到作业模板记录的 **extra_vars** 的格式。YAML 返回为保留格式和注释的 YAML，JSON 将返回为 JSON。

第 4 章 API 中的排序

为了让您易于使用，我们在整个指南中使用以下 URL：

```
https://<server name>/api/v2/groups/
```

流程

- 要指定 {{ model_verbose_name_plural }} 以特定顺序返回，请使用 **GET** 请求上的 **order_by** 查询字符串参数：

  ```
  https://<server name>/api/v2/model_verbose_name_plural?order_by={{ order_field }}
  ```

  - 为字段名称加上短划线(**-**)前缀，以反向排序：

    ```
    https://<server name>/api/v2/model_verbose_name_plural?order_by=-{{ order_field }}
    ```

  - 您可以使用逗号分隔字段名称（、、）来指定排序字段：

    ```
    https://<server name>/api/v2/model_verbose_name_plural?order_by={{ order_field }},some_other_field
    ```

第 5 章 使用搜索查询字符串参数

流程

- 使用 search query string 参数在模型的所有指定文本字段中执行非区分大小写的搜索：

```
https://<server name>/api/v2/model_verbose_name?search=findme
```

- 要跨相关字段搜索，请使用：

```
https://<server name>/api/v2/model_verbose_name?related__search=findme
```

第 6 章 在 API 中过滤

系统将集合识别为 "queryset"。您可以使用各种 operator 来过滤它。

流程

- 要查找包含名称 "foo" 的组，请使用：

```
http://<controller server name>/api/v2/groups/?name__contains=foo
```

- 要查找准确的匹配项，请使用：

```
http://<controller server name>/api/v2/groups/?name=foo
```

- 如果资源是整数类型，您必须将 __**int** 添加到结尾，以将字符串输入值转换为整数，如下所示：

```
http://<controller server name>/api/v2/arbitrary_resource/?x__int=5
```

- 您可以使用以下方法查询相关资源：

```
http://<controller server name>/api/v2/users/?first_name__icontains=kim
```

这会返回包含字符串"Kim"的名称的所有用户。

- 您还可以一次过滤多个字段：

```
http://<controller server name>/api/v2/groups/?
name__icontains=test&has_active_failures=false
```

这将找到所有包含名称"test"的组，但没有活跃故障。

其他资源

有关可用的运算符类型的更多信息，请参阅 QuerySet API 参考。

注意

您还可以监视 API，因为 UI 被用来查看如何过滤各种条件。

6.1. API 中的高级查询

您可以使用额外的查询字符串参数来过滤返回到与给定值匹配的结果列表。您只能使用数据库中存在的字段和关系进行过滤。确保指定的值中的任何特殊字符都是 URL 编码的。例如：

```
?field=value%20xyz
```

字段也可以跨越关系，仅适用于数据库中定义的字段和关系：

```
?other__field=value
```

要排除与特定条件匹配的结果，请为 field 参数添加 no __ 前缀：

```
?not__field=value
```

默认情况下，所有查询字符串过滤器都是 AND，因此仅返回与所有过滤器匹配的结果。要组合与多个条件中的任何一个匹配的结果，请为每个查询字符串参数添加前缀 **or__** ：

```
?or__field=value&or__field=othervalue
?or__not__field=value&or__field=othervalue
```

默认 AND 过滤同时应用于跨数据库关系过滤的每个相关对象。链过滤器会为每个相关对象单独应用过滤器。要使用它，请为查询字符串参数添加 **chain__** 前缀：

```
?chain__related__field=value&chain__related__field2=othervalue
?chain__not__related__field=value&chain__related__field2=othervalue
```

如果您将第一个查询写为 **?related*field=value&related*field2=othervalue**，它只会返回同时满足这两个条件的主对象。如使用链过滤器编写，它将返回与每个条件匹配的主对象的交集。

6.2. 字段查找

您可以通过将查找附加到字段名称，使用字段查找进行更高级的查询：

```
?field__lookup=value
```

支持以下字段查找：

- exact: Exact 匹配（如果没有指定，则默认查找）。
- iexact：完全不区分大小写的版本。
- contains: Field 包含值。
- icontains：包含不区分大小写的版本。
- startswith: Field 以值开头。
- istartswith：开头不区分大小写的版本.
- endwith: Field 以值结尾。
- iendswith：结尾不区分大小写的版本.
- 正则表达式：字段与给定的正则表达式匹配。
- iregex：正则表达式的不区分大小写版本。
- gt：而不是比较。
- gte ：大于或等于比较。
- lt: 少于比较。
- LTE：小于或等于比较。
- isnull ：检查给定字段或相关对象是否为 null；需要布尔值。

- 在 中：检查给定字段的值是否出现在提供的列表中；需要项目列表。
- 您可以为 true 指定布尔值为 **True** 或 **1**，**False** 或 **0** 代表 false （不区分大小写）。

例如，**?created__gte=2023-01-01** 提供了在 1/1/2023 后创建的项目列表。

您可以将 null 值指定为 **None** 或 **Null** （不区分大小写），但我们建议使用 **isnull** 查找来明确检查 null 值。

您可以将列表(**用于查询**)指定为以逗号分隔的值列表。根据请求的用户的访问权限级别通过查询字符串参数进行过滤：

- **role_level**: 要过滤的角色级别，如 **admin_role**

第 7 章 在 API 中使用分页

API 分页集合的响应。这意味着，虽然集合在每个 web 请求中可能包含几万或数百个对象，但出于 API 性能的原因，仅返回大量结果。

当您收到集合的结果时，会出现类似如下的内容：

```
{'count': 25, 'next': 'http://testserver/api/v2/some_resource?page=2', 'previous': None, 'results': [ ... ] }
```

流程

1. 请求"下一步"顺序 URL 给出的页面，以进入下一页。
2. 使用 **page_size=XX** 查询字符串参数更改每个请求返回的结果数。
   - **page_size** 的默认最大限值为 200，当用户尝试超过该值时（例如： **?page_size=1000** ）强制执行该限制。但是，您可以通过将 **/etc/tower/conf.d/<some file>.py** 中的值设置为 更高的内容来更改这个限制。例如，**MAX_PAGE_SIZE=1000**。
3. 使用 page 查询字符串参数来检索特定结果页面：

   ```
   http://<server name>/api/v2/model_verbose_name?page_size=100&page=2
   ```

与结果返回的前面的和以下链接会自动设置这些查询字符串参数。

不要请求大于 200 的页面大小。

用户界面使用较小的值来避免滚动。

第 8 章 访问资源

自动化控制器使用主键来访问单个资源对象。您可以通过命名 URL 功能使用特定于资源的人类可读标识符来访问自动化控制器资源。

以下示例显示了命名 URL 路径，您可以在其中访问没有辅助查询字符串的资源对象：

```
/api/v2/hosts/host_name++inv_name++org_name/
```

8.1. 配置设置

/api/v2/settings/named-url/: NAMED_URL_FORMATS 和 **NAMED_URL_GRAPH_NODES** 下有两个与 named-URL 相关的配置设置。

NAMED_URL_FORMATS 是所有可用命名 URL 标识符格式的只读键值对列表。下面显示了一个 **NAMED_URL_FORMATS** 示例：

```
"NAMED_URL_FORMATS": {
"organizations": "<name>",
"teams": "<name>++<organization.name>",
"credential_types": "<name>+<kind>",
"credentials": "<name>++<credential_type.name>+<credential_type.kind>++<organization.name>",
"notification_templates": "<name>++<organization.name>",
"job_templates": "<name>++<organization.name>",
"projects": "<name>++<organization.name>",
"inventories": "<name>++<organization.name>",
"hosts": "<name>++<inventory.name>++<organization.name>",
"groups": "<name>++<inventory.name>++<organization.name>",
"inventory_sources": "<name>++<inventory.name>++<organization.name>",
"inventory_scripts": "<name>++<organization.name>",
"instance_groups": "<name>",
"labels": "<name>++<organization.name>",
"workflow_job_templates": "<name>++<organization.name>",
"workflow_job_template_nodes": "<identifier>++<workflow_job_template.name>++
<organization.name>",
"applications": "<name>++<organization.name>",
"users": "<username>",
"instances": "<hostname>"
}
```

对于 **NAMED_URL_FORMATS** 中的每个项目，键是具有命名 URL 的资源的 API 名称。该值是一个字符串，指示如何为该资源形成人类可读的唯一标识符。**NAMED_URL_FORMATS** 仅列出可以具有命名 URL 的资源，任何没有列出的资源都没有命名 URL。如果资源可以具有命名 URL，则其对象必须具有一个 **named_url** 字段，该字段代表特定于对象的命名 URL。该字段仅在详情视图下可见，而不是列出视图。您可以使用准确生成的命名 URL 来访问指定的资源对象。这个对象及其相关 URL。例如，如果 **/api/v2/res_name/obj_slug/** 有效，**/api/v2/res_name/obj_slug/related_res_name/** 也有效。

NAMED_URL_FORMATS 指示足够编写人类可读的唯一标识符和命名 URL 本身。为了便于使用，可以具有命名 URL 的资源的每个对象都有 **名为_url 的** 相关字段，显示该对象的命名 URL。您可以复制并粘贴该字段，供您自己的自定义使用。如需更多信息，请参阅如果资源对象具有命名 URL，请参阅 API 浏览器的帮助文本。

您可以手动决定命名 URL 标签，例如使用 ID 5。要使用 **NAMED_URL_FORMATS** 来为这个特定资源对象编写命名 URL，请首先查找 **NAMED_URL_FORMATS** 的 labels 字段以获取标识符格式 <name> **++ <organization.name** > :

- URL 格式的第一部分是 < **name>**，这表示您可以在 **/api/v2/labels/5/** 中找到标签资源详情，并在返回的 JSON 中查找 **name** 字段。如果您有值为 **Foo** 的 **name** 字段，则唯一标识符的第一个部分为 **Foo**。

- 格式的第二个部分是双加号 ++。这是分隔唯一标识符不同部分的分隔符。将它们附加到唯一标识符中以获取 **Foo++**。

- 格式的第三个部分是 &lt **;organization.name** >，这表示该字段不在正在调查中的当前标签对象中，而是在标签对象指向的机构中。如格式所示，在当前返回的 JSON 的相关字段中查找机构。该字段可能不存在。如果存在，请按照该字段中指定的 URL （如 **/api/v2/organizations/3/** ）获取特定机构的详情，提取其 name 字段，如 "Default"，并将其附加到当前的唯一标识符中。因为 **<organizations.name** > 是格式的最后一部分，它生成以下命名 URL：
  /api/v2/labels/Foo++Default/。
  如果机构在标签对象详情的相关字段中不存在，请附加一个空字符串。这不会更改当前的标识符。因此，**Foo++** 成为最终的唯一标识符，生成的命名 URL 将变为 **/api/v2/labels/Foo++/**。

为命名 URL 生成唯一标识符的一个重要方面必须使用保留字符。因为标识符是 URL 的一部分，按 URL 标准对以下保留字符进行编码： **;/?:@=&[]**。例如，如果机构命名为 **;/?:@=&[]**，其唯一标识符应该是 **%3B%2F%3F%3F%3A%40%3D%26%5B%5D**。另一个特殊的保留字符是 **+**，它不由 URL 标准保留，而是使用命名 URL 来链接标识符的不同部分。它由 **[+]** 编码。例如，如果机构命名为 **[+]**，其唯一标识符为 **%5B[+]%5D**，其中原始 **[** 和 **]** 是编码的，则 **+** 转换为 **[+]**。

虽然您无法手动更改 **NAMED_URL_FORMATS**，但修改操作会随时间自动进行并扩展，以反映底层的资源修改和扩展。请参考您要使用命名 URL 功能的同一集群中的 **NAMED_URL_FORMATS**。

NAMED_URL_GRAPH_NODES 是另一个只读键值对列表，它公开用于管理命名 URL 的内部图形数据结构。这不是人类可读的，但必须用于以编程方式生成命名 URL。生成命名 URL 的示例脚本给出了任意资源对象的主键，其可以具有命名 URL，其使用 **NAMED_URL_GRAPH_NODES** 提供的信息，可在 GitHub 中找到。

8.2. 标识符格式协议

资源可以通过其唯一键来识别，这些键是资源字段的元组。每个资源都可以保证将其主键号作为唯一键，但可能存在许多其他唯一的键。资源可以生成标识符格式，因此如果它至少有一个满足以下规则的唯一键，则具有命名 URL：

1. 键必须只包含 **name** 字段或文本字段，并带有有限数量的选项（如凭证类型资源的 **kind** 字段）。

2. 唯一允许的特殊字段会破坏上述规则是与自身以外的资源相关的多对一相关字段，它也允许具有 slug。

如果有资源 **Foo** 和 **Bar**，则 **Foo** 和 **Bar** 都包含一个 name 字段和一个 choice 字段，它只能具有 "yes" 或 "no"。此外，资源 **Foo** 具有一个与 **Bar** 相关的多对一字段（例如 **fk** ）。**foo** 具有唯一的键元组(**名称**、选择、**fk**)，而 **Bar** 具有唯一的键元组(**名称**，选择)。**bar** 可以具有命名 URL，因为它满足前面的第一条规则。foo 也可以具有命名 URL，即使它会破坏第一个规则，但额外的字段破坏规则号为 **fk** 字段，它是一个与 **Bar** 和 **Bar** 相关的许多与一对一相关的字段。

对于满足规则编号 1 的资源，其人类可读的唯一标识符组合是外键字段的组合，用 + 分隔。特别是，上例中的资源 Bar 具有 slug 格式 < name>+<choice>。请注意，字段顺序在 slug 格式的问题，如果存

在，则名称字段始终首先出现，后跟字段名称的字典顺序排列的其余字段。例如，如果 Bar 也具有满足规则一的 a_choice 字段，并且唯一键变成 （名称、选择、a_choice ），其 slug 格式将变为 <name >< a_choice><choice>。

对于满足规则编号 2 的资源，如果通过额外的外键字段进行追踪，则结果是用于标识该资源对象的资源树。为了生成标识符格式，追溯树中的每个资源都会使用外键以外的所有字段生成自己的独立格式部分。最后，所有部分都由 ++ 按照以下顺序合并：

- 将单机格式作为第一个标识符部分设置。
- 为每个资源递归生成唯一标识符。底层资源指向使用外键（追溯 树节点的子代）。
- 将生成的唯一标识符视为标识符组件的其余部分。按照对应外键的字典顺序对它们进行排序。
- 使用 ++ 组合所有组件来生成最终标识符格式。

当为资源 Foo 生成标识符格式时，自动化控制器会生成独立格式，< name>+<choice> 用于 Foo 和 < fk.name>+<fk.choice > 用于 Bar，然后将它们合并为 < name><choice>+<fk.name>+ <fk.choice>。

根据给定的标识符格式生成标识符时，外键可能指向任何位置。在这种情况下，自动化控制器替换了与该资源对应的格式部分，外部键应指向空字符串。例如，如果 Foo 对象的名称为 ="alice"，则 选择 ="yes"， 但 fk field = None，其生成的标识符为 alice+yes++。

第 9 章 只读字段

REST API 中的某些字段被标记为只读。

它们通常包括资源的 URL、ID 和偶尔一些内部字段。例如，每个对象的 'created_by' 属性指示哪个用户创建了资源，您无法编辑它。如果您发布了一些值，并且注意到它们没有更改，这些字段可能是只读的。

第 10 章 在 API 中进行身份验证

您可以在 API 中使用以下验证方法：

- 会话身份验证
- 基本身份验证（Basic authentication）
- OAuth 2 令牌身份验证
- 单点登录身份验证

自动化控制器专为机构设计，通过可视化仪表板来集中和控制其自动化，以进行开箱即用的控制，同时提供 REST API 以在更深入的级别上与其他工具集成。自动化控制器支持多种身份验证方法，可以轻松地将自动化控制器嵌入到现有工具和流程中。这可确保适当的人员可以访问其资源。

10.1. 使用会话身份验证

在直接登录到自动化控制器的 API 或 UI 时，您可以使用会话身份验证来手动创建资源，如清单、项目和作业模板，并在浏览器中启动作业。使用这个方法，您可以长时间保持登录，而不仅仅是该 HTTP 请求。例如，当在浏览器中（如 Chrome 或 Mozilla Firefox）中浏览 API 或 UI 时。当用户登录时，会创建一个会话 Cookie，这意味着当导航到自动化控制器中的不同页面时，它们可以保持登录。下图代表会话中客户端和服务器之间发生的通信：

使用 curl 工具查看登录到自动化控制器时发生的活动。

流程

1. 使用 GET 进入 /api/login/ 端点来获取 csrftoken cookie：

```
curl -k -c - https://<controller-host>/api/login/

localhost      FALSE   /      FALSE   0   csrftoken
AswSFn5p1qQvaX4KoRZN6A5yer0Pq0VG2cXMTzZnzuhaY0L4tiidYqwf5PXZckuj
```

2. POST 到 /api/login/ 端点，用户名、密码和 X-CSRFToken=<token-value> ：

```
curl -X POST -H 'Content-Type: application/x-www-form-urlencoded' \
--referer https://<awx-host>/api/login/ \
-H 'X-CSRFToken:
K580zVVm0rWX8pmNylz5ygTPamgUJxifrdJY0UDtMMoOis5Q1UOxRmV9918BUBIN' \
--data 'username=root&password=reverse' \
--cookie
'csrftoken=K580zVVm0rWX8pmNylz5ygTPamgUJxifrdJY0UDtMMoOis5Q1UOxRmV991
8BUBIN' \
https://<awx-host>/api/login/ -k -D - -o /dev/null
```

当您在浏览器中登录到 UI 或 API 时，所有这些都由自动化控制器完成，且只能在浏览器中进行身份验证时使用它。有关与自动化控制器的编程集成，请参阅 OAuth2 令牌身份验证。

以下是典型的响应示例：

```
Server: nginx
Date: <current date>
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: keep-alive
Location: /accounts/profile/
X-API-Session-Cookie-Name: awx_sessionid
Expires: <date>
Cache-Control: max-age=0, no-cache, no-store, must-revalidate, private
Vary: Cookie, Accept-Language, Origin
Session-Timeout: 1800
Content-Language: en
X-API-Total-Time: 0.377s
X-API-Request-Id: 700826696425433fb0c8807cd40c00a0
Access-Control-Expose-Headers: X-API-Request-Id
Set-Cookie: userLoggedIn=true; Path=/
Set-Cookie: current_user=<user cookie data>; Path=/
Set-Cookie: csrftoken=<csrftoken>; Path=/; SameSite=Lax
Set-Cookie: awx_sessionid=<your session id>; expires=<date>; HttpOnly; Max-Age=1800;
Path=/; SameSite=Lax
Strict-Transport-Security: max-age=15768000
```

当使用此方法成功验证用户时，服务器会使用名为 X-API-Session-Cookie-Name 的标头进行响应，指示会话 Cookie 的配置名称。默认值为 awx_session_id，稍后您可以在 Set-Cookie 标头中看到。

注意

您可以通过在 SESSION_COOKIE_AGE 参数中指定会话过期时间来更改会话过期时间。如需更多信息，请参阅 使用会话限制。

10.2. 基本身份验证（BASIC AUTHENTICATION）

基本身份验证是无状态的，因此您必须通过 Authorization 标头发送 base64 编码的用户名和密码以及每个请求。您可以使用它来来自 curl 请求、python 脚本或单独对 API 的请求的 API 调用。我们建议 OAuth 2 令牌身份验证在可能的情况下访问 API。

以下是使用 curl 进行基本身份验证的示例：

```
# the --user flag adds this Authorization header for us
curl -X GET --user 'user:password' https://<controller-host>/api/v2/credentials -k -L
```

其他资源

有关基本身份验证的更多信息，请参阅 'Basic' HTTP Authentication Scheme。

禁用基本身份验证

您可以为安全禁用基本身份验证。

流程

1. 在导航面板中，选择 Settings。
2. 从 系统 选项列表中选择 Miscellaneous Authentication settings。
3. 禁用 选项以启用 HTTP Basic Auth。

10.3. OAUTH 2 令牌身份验证

OAuth (Open Authorization)是基于令牌的身份验证和授权的公开标准。OAuth 2 身份验证通常用于以编程方式与自动化控制器 API 交互。与基本身份验证类似，您可以通过 Authorization 标头为每个 API 请求提供一个 OAuth 2 令牌。与基本身份验证不同，OAuth 2 令牌有一个可配置的超时，并可范围。令牌具有可配置的过期时间，如果需要，可以为一个用户或管理员轻松撤销整个自动化控制器时间。您可以

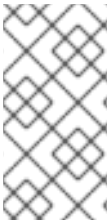
使用 revoke_oauth2_tokens 管理命令进行此操作，也可以使用 API，如 Revoke an access token 所述。

在自动化控制器中获取 OAuth2 访问令牌的不同方法包括：

- 个人访问令牌(PAT)
- 应用程序令牌：密码授权类型
- 应用程序令牌：隐性授权类型
- 应用程序令牌：授权代码授权类型

用户需要在 API 或自动化控制器 UI 的 Users > Tokens 选项卡中创建 OAuth 2 令牌。有关通过 UI 创建令牌的更多信息，请参阅 用户 - 令牌。

在本例中，使用 PAT 方法在 API 中创建令牌。创建后，您可以设置范围。



注意

您可以在系统范围内配置令牌的过期时间。如需更多信息，请参阅 将 OAuth 2 令牌系统用于个人访问令牌。

令牌身份验证最适合用于自动化控制器 API 的任何编程使用，如 Python 脚本或 curl 等工具。

curl 示例

```
curl -u user:password -k -X POST https://<controller-host>/api/v2/tokens/
```

此调用返回 JSON 数据，如下所示：

您可以使用 token 属性的值为自动化控制器资源执行 GET 请求，如 Hosts ：

```
curl -k -X POST \
  -H “Content-Type: application/json”
  -H “Authorization: Bearer <oauth2-token-value>” \
  https://<controller-host>/api/v2/hosts/
```

您还可以通过向要启动的作业模板发布 POST 来运行作业：

```
curl -k -X POST \
  -H "Authorization: Bearer <oauth2-token-value>" \
  -H "Content-Type: application/json" \
  --data '{"limit" : "ansible"}' \
  https://<controller-host>/api/v2/job_templates/14/launch/
```

Python 示例

awxkit 是一个开源工具，可以轻松地使用 HTTP 请求来访问自动化控制器 API。您可以使用 awxkit login 命令代表您使用 awxkit 登录命令获得 PAT。如需更多信息，请参阅 AWX 命令行界面。

如果您需要编写自定义请求，您可以使用 Python 库请求编写 Python 脚本，如下例所示：

```
import requests
oauth2_token_value = 'y1Q8ye4hPvT61aQq63Da6N1C25jiA'   # your token value from
controller
url = 'https://<controller-host>/api/v2/users/'
payload = {}
headers = {'Authorization': 'Bearer ' + oauth2_token_value,}

# makes request to controller user endpoint
response = requests.request('GET', url, headers=headers, data=payload,
allow_redirects=False, verify=False)

# prints json returned from controller with formatting
print(json.dumps(response.json(), indent=4, sort_keys=True))
```

其他资源

有关获取 OAuth2 访问令牌以及如何在外部应用上下文中使用 OAuth 2 的更多信息，请参阅 *自动化控*

*制器管理指南中的基于令牌的身份验证。*https://docs.redhat.com/en/documentation/red_hat_ansible_automation_platform/2.4/html-single/automation_controller_administration_guide/index#assembly-controller-token-based-authentication

启用外部用户创建 OAuth 2 令牌

默认情况下，通过单点登录创建的外部用户无法为安全目的生成 OAuth 令牌。

流程

1. 在导航面板中，选择 Settings。
2. 从 系统 选项列表中选择 Miscellaneous Authentication settings。
3. 启用 选项，以允许外部用户创建 OAuth2 令牌。

10.4. 单点登录身份验证

单点登录(SSO)身份验证方法与其他方法不同，因为用户的身份验证发生在自动化控制器外部，如 Google SSO、Microsoft Azure SSO、SAML 或 GitHub。例如，在 GitHub SSO 中，GitHub 是单一来源，它根据您提供自动化控制器的用户名和密码验证您的身份。

您可以使用带有中央身份提供程序的大型机构中的自动化控制器来配置 SSO 身份验证。在自动化控制器中配置了 SSO 方法后，登录屏幕上就提供了该 SSO 的选项。如果点该选项，它会将您重定向到身份提供程序，本例中为 GitHub，其中为您提供了您的凭证。如果身份提供程序验证您是否成功，自动化控制器将使用户链接到您的 GitHub 用户（如果这是您第一次使用此 SSO 方法登录）并登录。

其他资源

有关各种支持的 SSO 身份验证方法，*请参阅自动控制器管理指南中的* 设置社交身份验证 和设置企业身份验证。