8/6/2022, 8/11, 8/13, 8/15, 8/19,

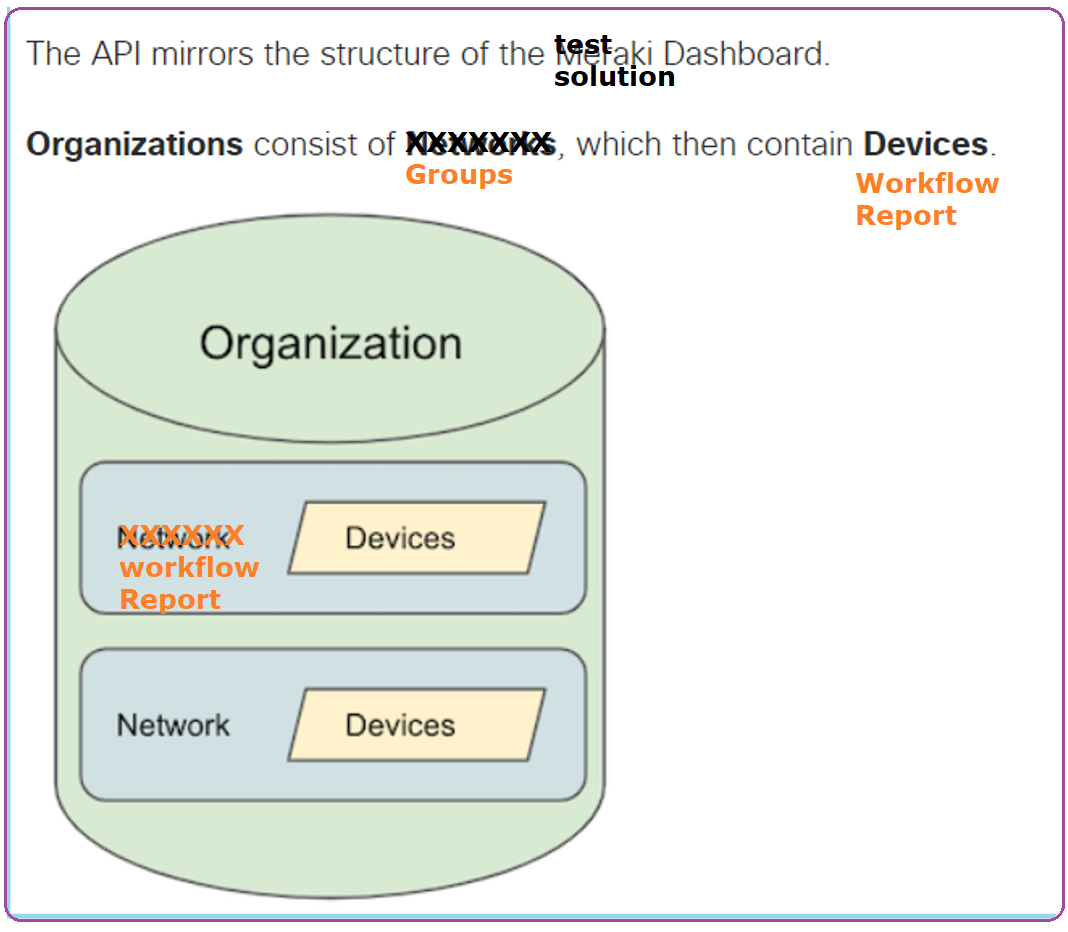
1. venders
2. Test tools
3. Test management tools
4. Test automation pytest
5. Selenium
6. Projects
7. Figure interconnections between user/device/workflow/token/test/projects(team)
8. TBD

##### **API hierarchy**

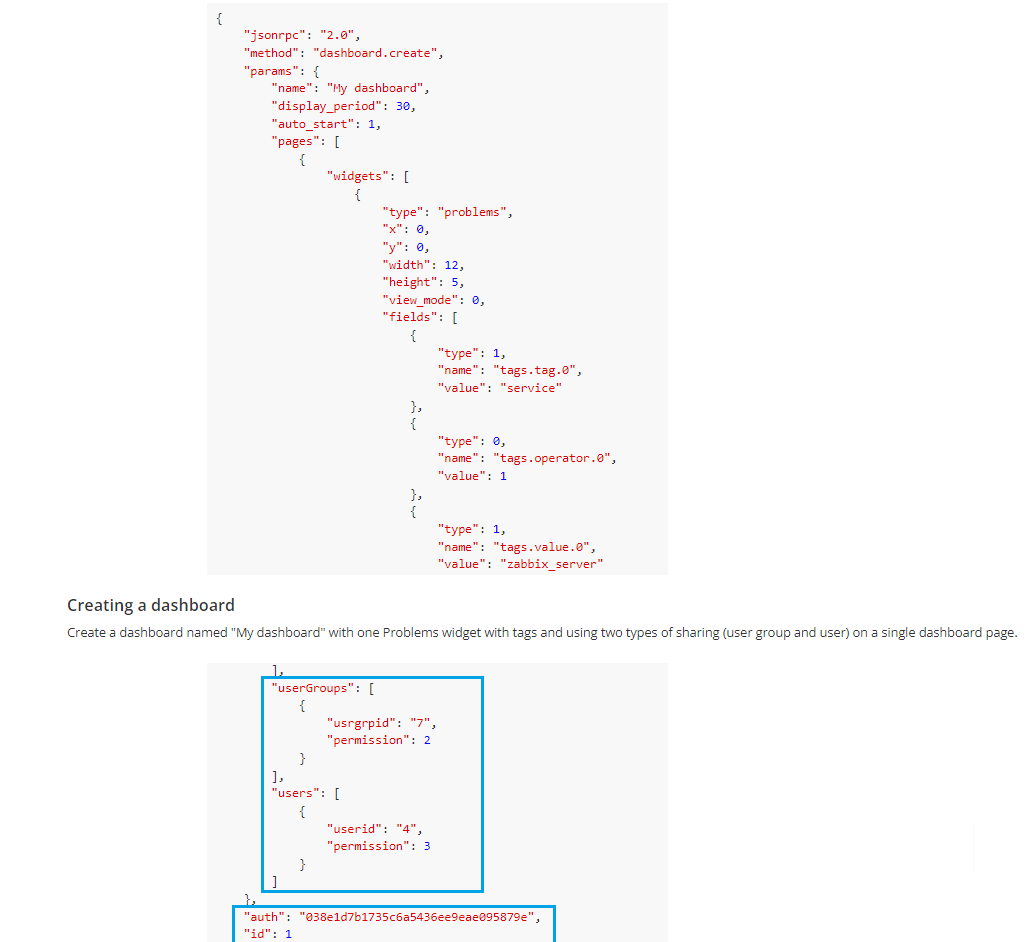
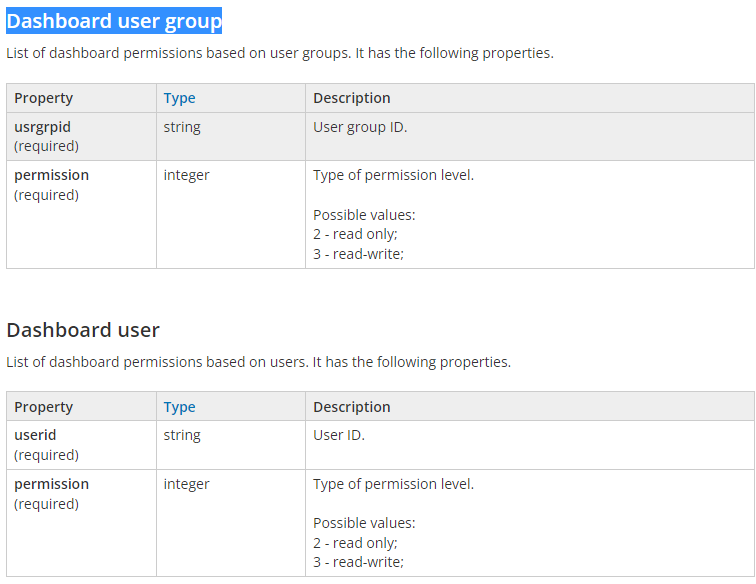
There are two basic types of dashboard administrators:  **Group(team) administrators within an organizations (organization ID, Group ID)**.

* **Group administrators – Group ID** have access to and their devices.
  + **User within a group**: user has read and write permission for all the resources within the group; only read rights for resources outside its own group
  + **Superuser : has read/write rights for resources across groups and can create/edit/delete work\_projects. (normally project manager, can create projects)…**
* The Dashboard API has evolved significantly, providing **about 100 endpoints** to manage **workflow, devices and reports**.
* The API documentation, Postman collection, and Python library will remain synced and up-to-date with improved navigation and features.

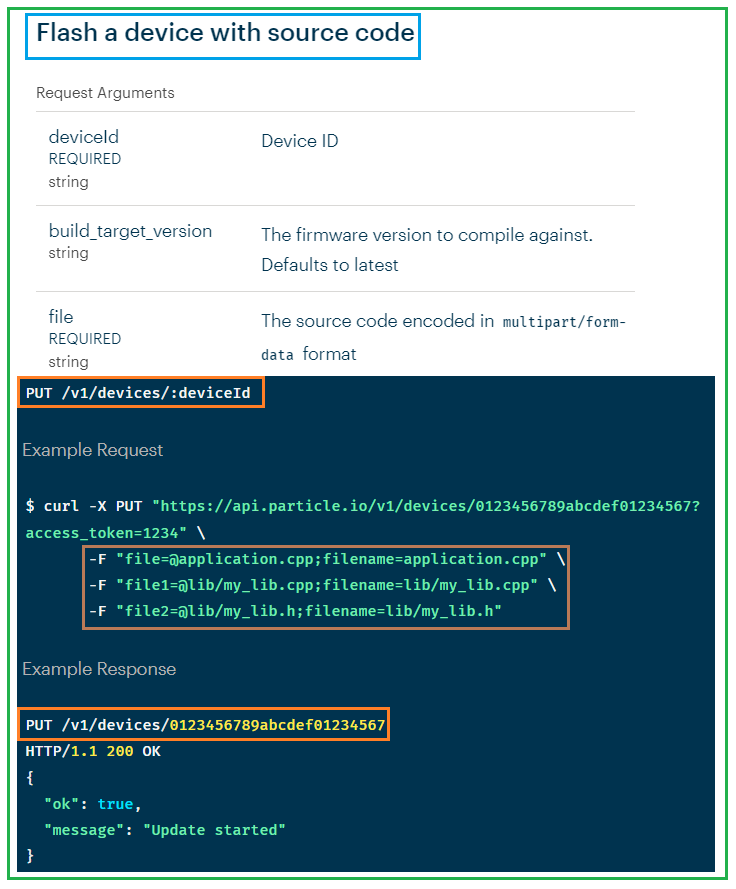
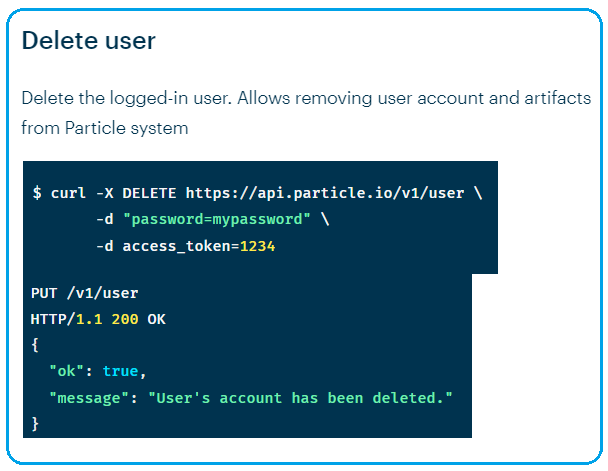
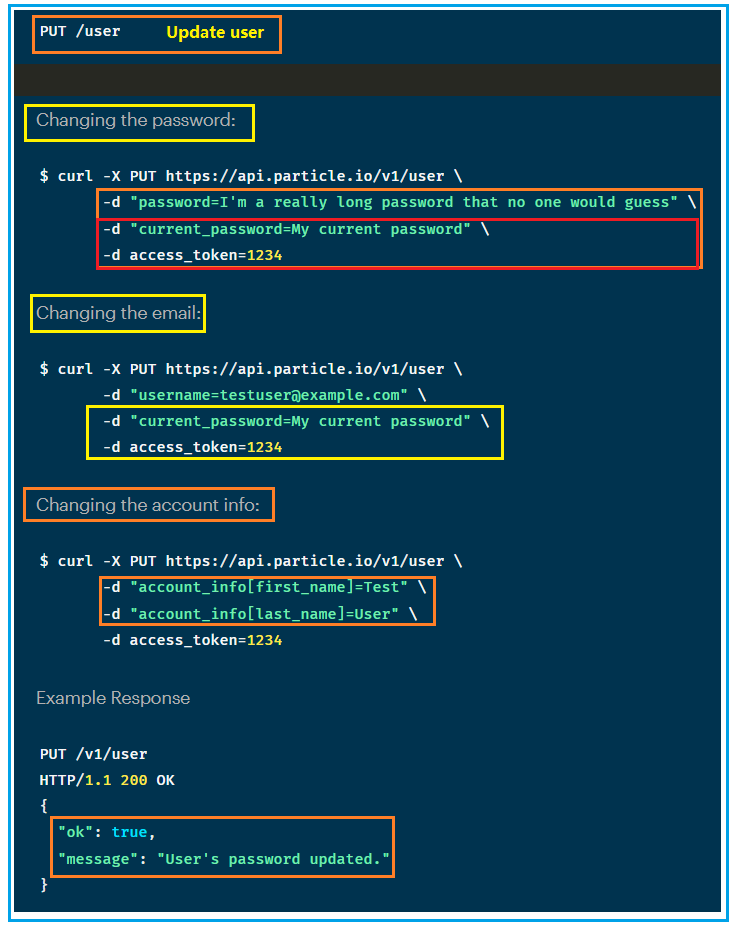
1. **Base API**
2. [**https://api.litepoint.com/api/v1**](https://api.litepoint.com/api/v1)

****

1. **SDK**
2. SDKs
3. the custom  [Python library](https://developer.cisco.com/meraki/api-v1/#python) will be the recommended SDK for simplified API scripting.
4. venders
5. <https://www.zabbix.com/documentation/current/en/manual/api/reference/dashboard/create>
6. <https://docs.particle.io/reference/cloud-apis/api/#list-devices-in-a-product>







<https://www.zabbix.com/documentation/current/en/manual/web_interface/frontend_sections/administration>

1. <https://docs.particle.io/reference/cloud-apis/api/#user>
2. Test tools
3. Test management tools
4. <https://blog.testproject.io/2021/02/15/built-in-cloud-reporting-tool-for-pytest-and-unittest/>
5. <https://www.eswcompany.com/python-test-tools-should-you-be-using/>
6. <https://www.fugue.co/blog/blog-post-creating-an-automated-cloud-infrastructure-testing-tool-with-terraform-and-pytest>
7. Test automation pytest

# [How can I write Python code to support both Windows and Linux](https://stackoverflow.com/questions/58670743/how-can-i-write-python-code-to-support-both-windows-and-linux)

# <https://stackoverflow.com/questions/58670743/how-can-i-write-python-code-to-support-both-windows-and-linux>

# how to maintain one python code base in both linux and windows

1. Selenium
2. Projects
3. User

* User/team/
* During internal testing/customer feed back, noticed that user/credential with group to manage devices is not efficient when devices are shared
* Recommend add API team, which is project based.
* Team/projects, within a project or team, all users have same access rights to the devices, tests, and work\_flows.
* Project manager is created as superuser. Who can create team, add users to the team, add resources/ devices etc.
* ;;
* Within a group, superuser vs user
* Superuser can assign devices/test/test\_flows ownership to other user
* User can only transfer ownership to other user.
* When user creates device, work\_flow, report; they can set the access to read only or read and write. Superuser can overwrite the access property.

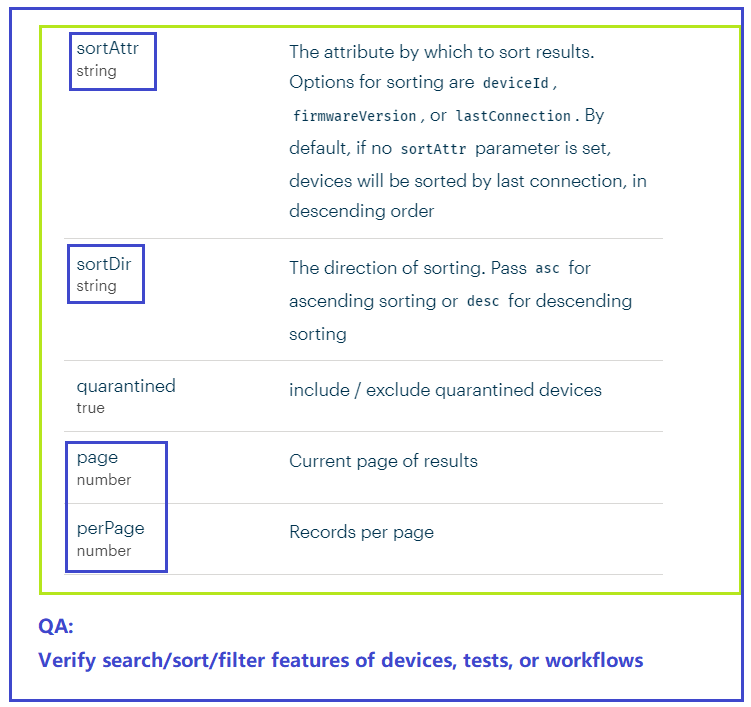
1. Devices

* List all devices that are part of a team(project) or user. Results are paginated, by default returns 25 device records per page
* Curl [https://server:port/v1/teams/{teamID}/devices?access\_token=1234](https://server:port/v1/teams/%7bteamID%7d/devices?access_token=1234)
* **Search/Filter/Sort**
* Show devices from a project and specific user

Curl https:/server:port/v1/devices?team=teamID&user=userID





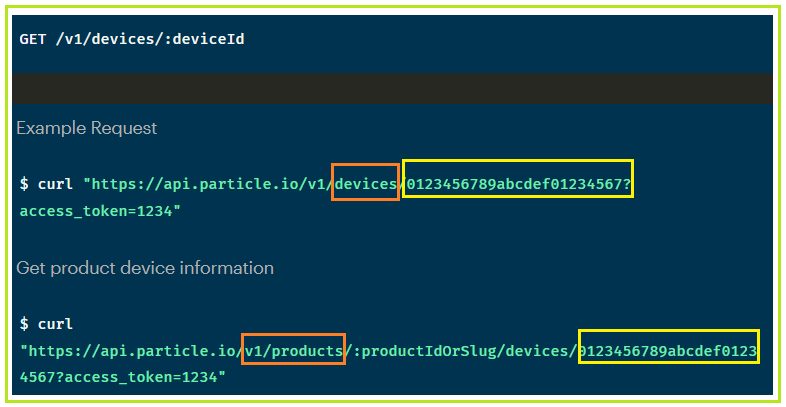


* Verify **inter-connect** between APIs: same data should be consistent via different API calls.

V1/team/devices: shows all the teams with their corresponding assigned devices

/api/v1/devices: show devices and which team they are assigned to if any

So if a device is removed/updated from devices, the changes should automatically reflected from its assigned teams.



--

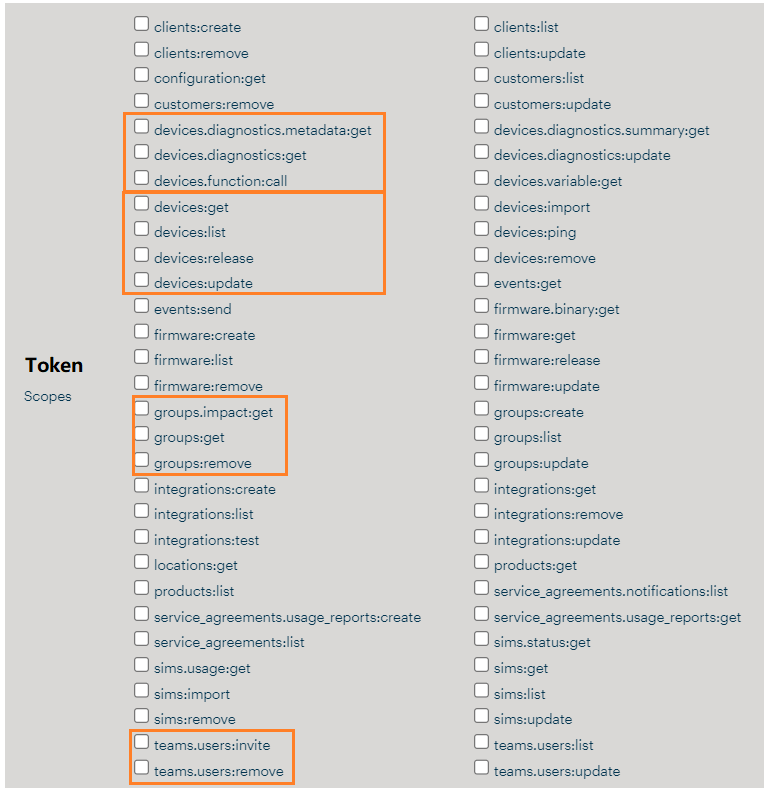
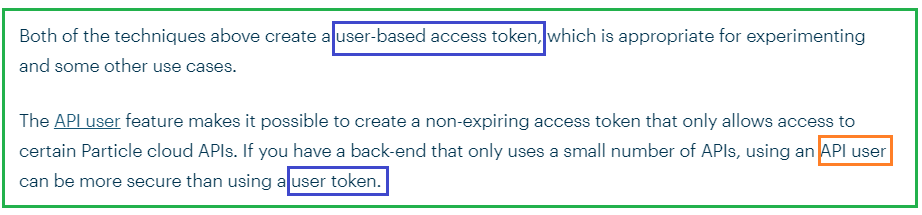
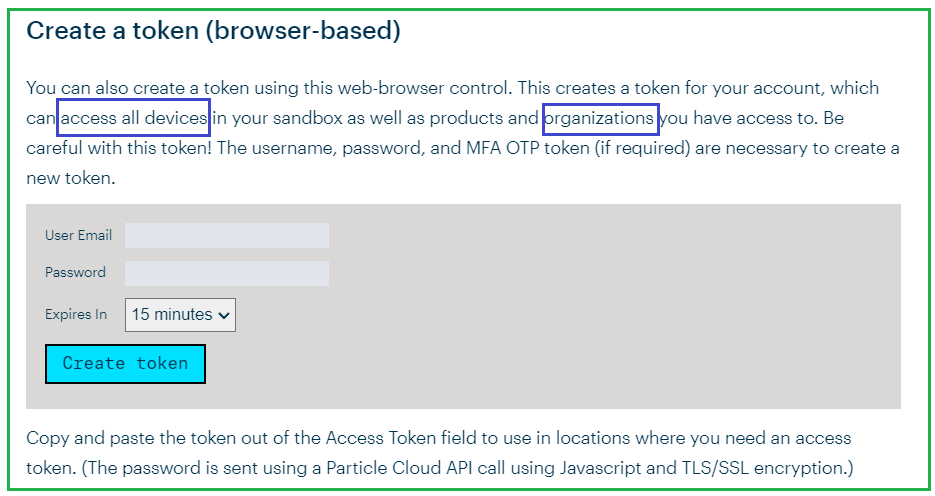


* **Cross-check API calls**

****

1. Token

* Token serves mainly for API access as an identity metric. It carries the same rights according to its user.
* Our token is user based, not api call based.
* <https://docs.particle.io/reference/cloud-apis/access-tokens/>



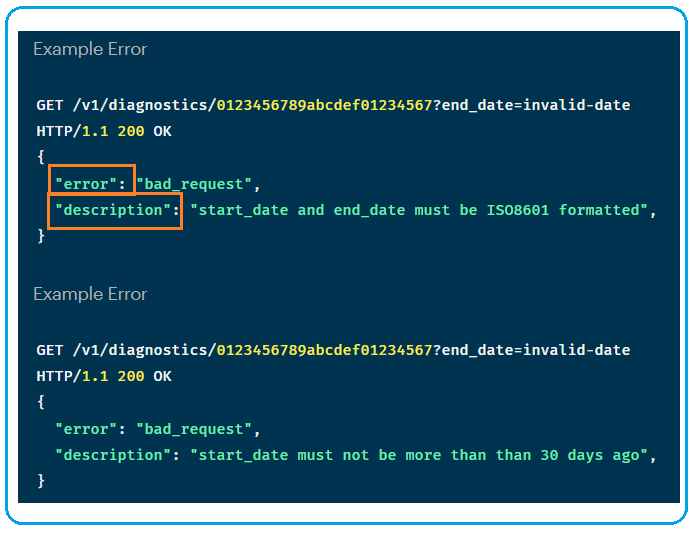
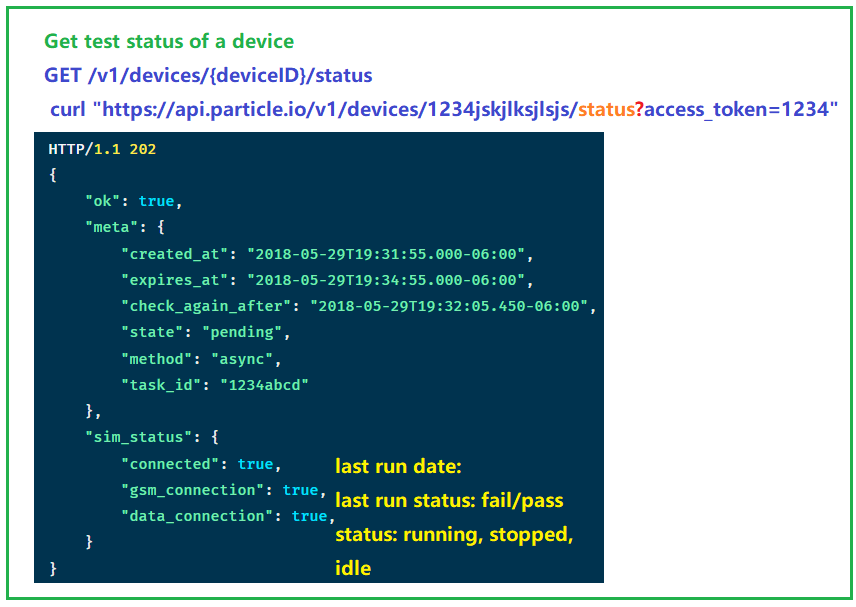
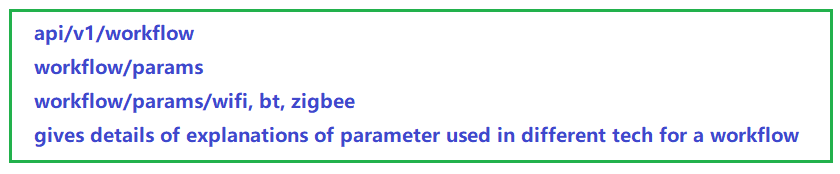
1. Error Checking of response

* A good source as resume material.

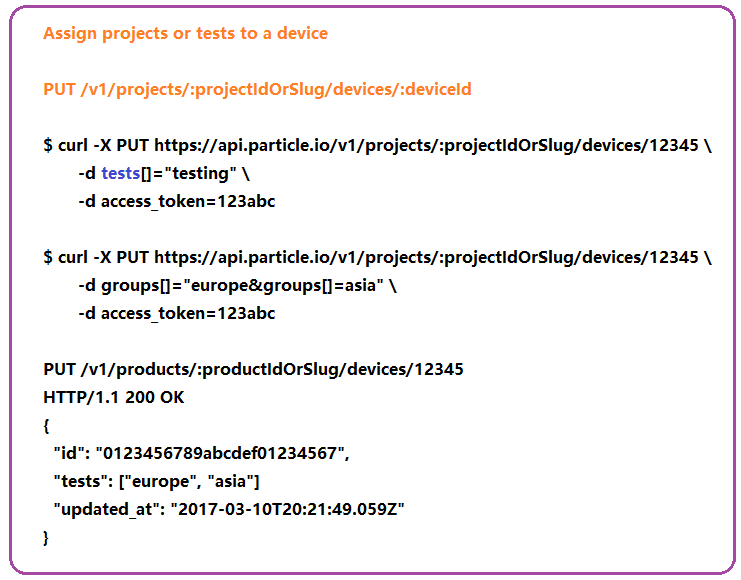
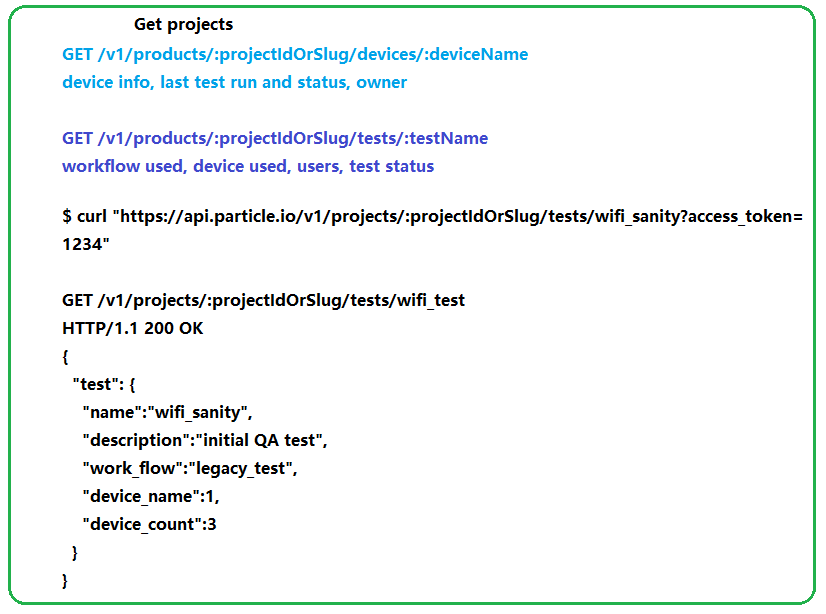
1. Api/test, api/work\_flow, api\_report

* **Get historical tests run by a device**



****

* **/Project, /group**

****

1. A
2. Figure interconnections between user/device/workflow/token/test/projects(team)
3. Understand **JIRA server implementation** (on premise):

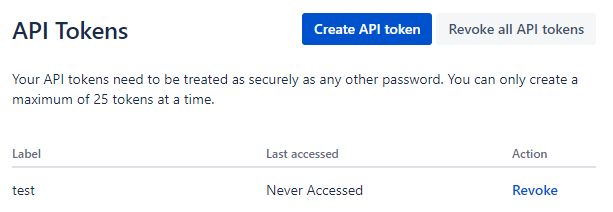
Jira is available in two versions, a Cloud version and a **Server (on-premise) version**. The number of add-ons for Jira Cloud is very limited and aims at smaller teams, Jira Server has many more add-ons and aims at larger organizations.

1. JIRA as an example

[**https://jira.litepoint.com**/projects/SYSTEMTEST/issues/SYSTEMTEST-4716?**filter**=allissues](https://jira.litepoint.com/projects/SYSTEMTEST/issues/SYSTEMTEST-4716?filter=allissues)

[**https://lpbrcm.atlassian.net**/jira/software/c/projects/B89/issues/?**jql**=project%20IN%20(%22B89%22)%20**ORDER**%20BY%20**created**%20DESC](https://lpbrcm.atlassian.net/jira/software/c/projects/B89/issues/?jql=project%20IN%20(%22B89%22)%20ORDER%20BY%20created%20DESC)

* It is quite an extended API and gives you the opportunity to write your own scripts extending the Jira functionality.
* [https://dzone.com/articles/**how-to-use-the-jira-api**](https://dzone.com/articles/how-to-use-the-jira-api)
* https://dzone.com/articles/a-jira-tutorial-for-software-developers-get-the-mo
* <https://www.atlassian.com/software/jira/free>
* starcsi@atlassian.net
* token(test): PS3W9y9rJlighnckftthBAF0 (len=24)



<https://confluence.atlassian.com/alldoc/atlassian-documentation-32243719.html>

<https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/>

The Jira REST API enables you to **interact with Jira programmatically**. Use this API to [**build** **apps**](https://developer.atlassian.com/cloud/jira/platform/integrating-with-jira-cloud/), **script interactions**/ **automation** with Jira, or develop any **other type of integration**

## **Permissions**

https://support.atlassian.com/jira-cloud-administration/docs/manage-project-permissions/?permissionViolation=true

Most operations in this API require permissions. The calling user must have the required permissions for an operation to use it. **Permission scheme is created by IT admin**. When users are created, their permission level is determined based on their role.

* Users bounded to a group
* Super-users or project admin (right across multiple groups)
* One employee could have multiple level of permissions.
* Users and its associated token is determined by permission scheme depending on user’s role within the organization or group.

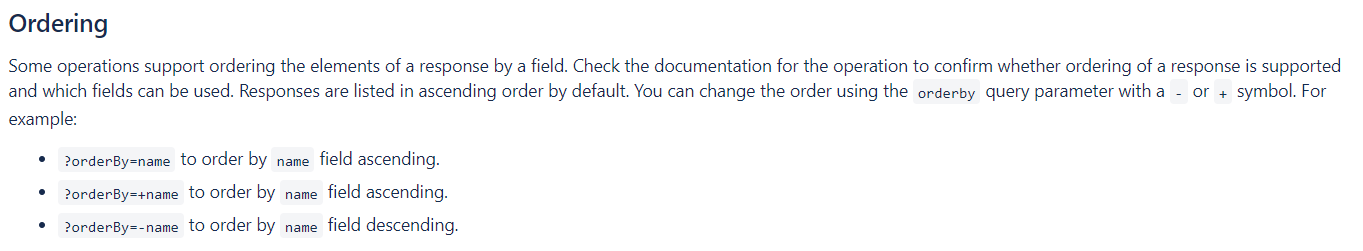
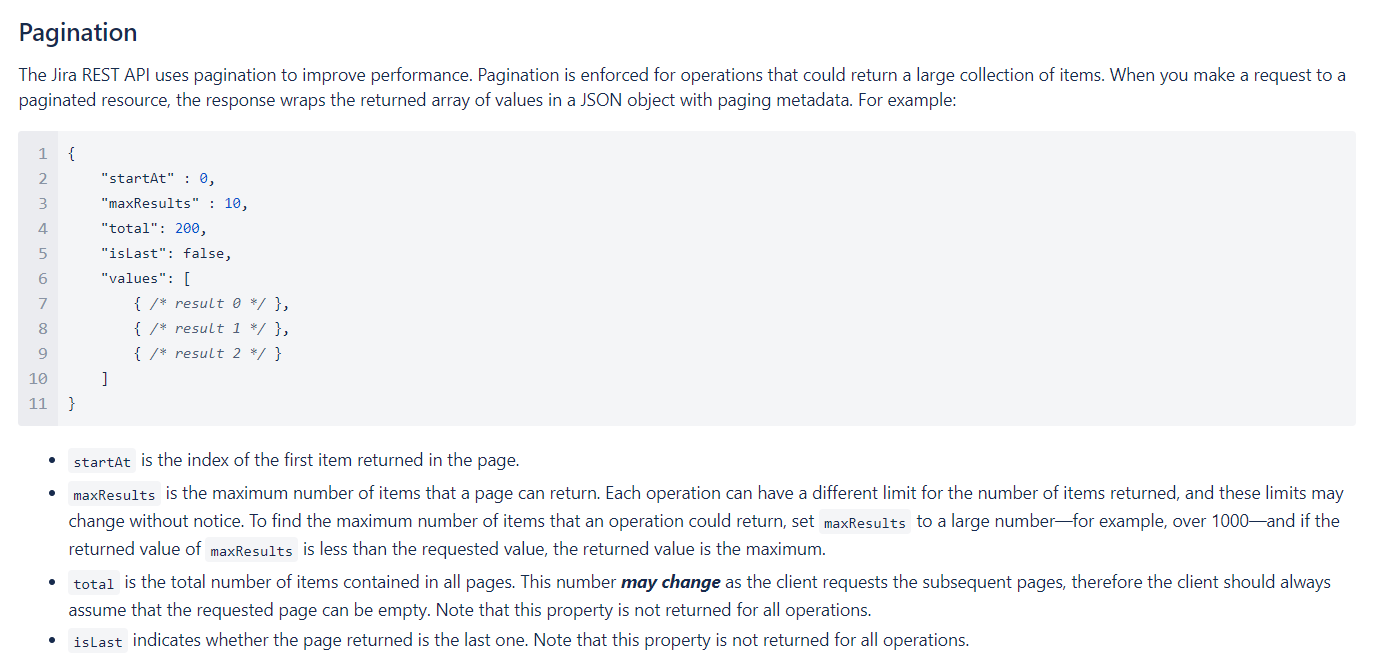
## **Create a permission scheme**

IT can update your permission scheme by adding user to multiple groups or projects or roles so certain rights can be granted automatically.

1. Add **users**, **groups**, and **roles** to the scheme and grant their project permissions.
2. Associate the scheme with the projects that should use it.

A permission can be granted to a **group**, **project role**, or issue role that **the user is a member of, or granted directly to a user**.

* IT admin of the server within an **organization**.
* User within a group has r-w or read-only access to resources within a **group**. (Group admin, corporate IT manage)
* Project admin (program manager ) create a project, assign devices/workflow, users and other resources to the **project**. Users in the project has r-w or read-only to resources within a project. Except adding/removing new devices or users.



<https://developer.atlassian.com/cloud/jira/platform/rest/v2/api-group-groups/#api-rest-api-2-group-bulk-get>

##### **LP APIs**

lp APIs provide control of the **test solution in a programmable way**, enabling actions that may not be possible with the dashboard, or proving more granular control. APIs are **RESTful APIs using HTTPS** for transport and **JSON** for object serialization.

Through APIs, users can **automate managing projects/testFlows/devices**, **monitor test/device status**, and **build additional solutions on top of the lp APIs**.

**API keys/tokens are tied to a specific** user **account**.

**superuser** has **administrative** access: create/manage projects,

#### Server Data Segregation

**User data on Meraki servers is segregated based on user permissions**. **Each user account is authenticated based on organization membership**, meaning that each user only has access to information tied to the organizations they have been added to as users. **Organization administrators add users to their own organizations**, and those users set their own username and secure password. That user is then tied to that **organization’s unique ID**, and is then only able to make requests to Meraki servers for data scoped to their authorized organization IDs.

Additionally, **the Meraki development teams have separate servers for development and production, so Meraki never uses live customer data for testing or development**. Meraki user data is never accessible to other users or subject to development changes.

###### **API Key/Token**

API keys/tokens are tied to **the access of the user** who created them.  Programmatic access should only be granted to those entities who you trust to work within the organizations they are assigned to. Because **API keys/Token are tied to accounts** within an organization.

1. TBD