2/26/22,3/1/22, 5/10

1. **How my UI** should look like
2. User tab

* Log in and change passwd
* Contact info: email, group/org id and name

1. How other company’s API, API call and API response look like
2. **Overall Structure**

All tests are run by LP Testing Application/solution on the test devices reflecting real end-user quality of experience. By combining LP connectivity Benchmarking Solution with LP workflow generator, analysis and reporting. you can create an **end-to-end automated data processing environment** from the collection of measurement data to analytics and reporting.

1. **Client: device and tester**
2. Client\_side: install and run test\_program as Command Line Tool (IQfactrun\_Console)
3. Install an agent

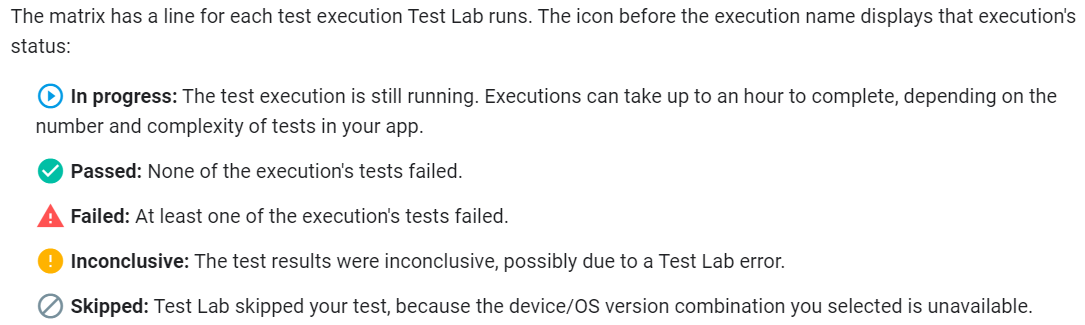
* Monitor status of test\_program: running, idling or hang
* Updates the test running status to server, # of failures, # of passes.
* Agent can also communicate test\_program, trigger the program to run or quit.

1. Upload test results/work flows to server, download the analysis/summary.
2. ***Server***
3. Manage device/tester/user **(UI/API focus 1)**
4. **Create/modify work/test flows (UI focus 2 and API focus)**
5. Agent monitoring test status during test running
6. Provide quick test summary/analysis (**UI and API: 3**)
7. OAuth Authentication for both end users and web/mobile APP (**UI and API: 4**).
8. ***Performance Test***
9. Upload/download test
10. Monitor local PC/Server: memory Usage, CPU, and data written to storage.
11. ***Run from desktop***
12. Download and install the litepoint SDK

* Include test program IQfactrun\_Console, IQwizard for flow, agent, we can say agent is part of the IQfactrun\_Console.

1. Log in (optional)
2. Run the test

* UI: Let’s not use UI to run the test. Use UI only for monitoring and status purpose. first download workflow from server to workflow\_path(**public drive on line like iqfiler**), PC\_IP, device\_id, tester\_id; server pass the parameters to agent so agent can kick off test.
* server
* Configure the tests, # of devices, test\_flows, # of runs
* IQfactrun\_console test\_flow options
* You can use shell scripts or batch files to automate tests.
* Dashboard shows the in-progress test matrix;





1. Investigate test results

* When the test starts, you receive a link to the test results page.
* After tests have run, you can review test summary in the dashboard.
* I think for now just the summary (# passed, failed, and their test case names in a matrix form; NOT raw logs since I don’t think it is the job of agent to upload big log files, or screenshot to server. Client has to manually upload them to server for analysis.

1. ***Endpoints***
2. Base URL: <https://server:port/api/v1>
3. Basic Path**: /organization/{organization\_id}/group/{group\_id}/**user/{user\_id}/device/{device\_id}

**/organization/{organization\_id}/group/{group\_id}/user**/{user\_id}/

* /user/{user\_id}/**device**
* /user/{user\_id}/**workflow** (how to differentiate flows, with ID is inconvenient)

1. User/{user\_id}

* Group\_name and ID
* User\_name and ID
* Contact info: email, phone
* Date registered
* Method
* Get
* Post
* Put

1. user/device/{device\_id}

* Device model name, device ID
* IP
* owner
* OS version, FW version
* Test\_status: idle, running, stopped
* Last/current run test statistics, date, pass/fail
* Method
* /devices/{device\_id}/tests
* Shows all previous test statistics
* /devices/{device\_id}/tests/{test\_id}
* Show detailed test status of a specific test

1. user/tests

* test\_name, test\_id
* device, tester
* user
* pass/fail
* time test completed
* workflow\_name/id

1. User/Workflows

* Workflow ID, name
* Description
* Coverage
* Created @
* Last Modified
* Last tested @
* Details test matrix in json

1. Analysis/analytics

* user/reports
* /reports/{report\_id}

1. Group/users
2. Authorization

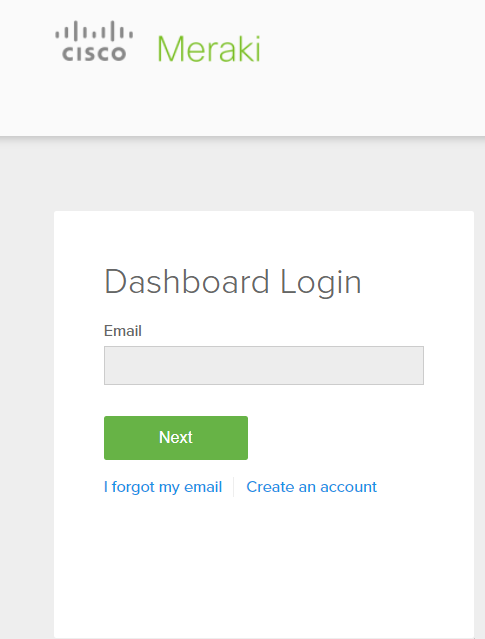
* https://developer.cisco.com/meraki/api-v1/#!authorization/authorization-with-bearer-token-beta

1. /organizations

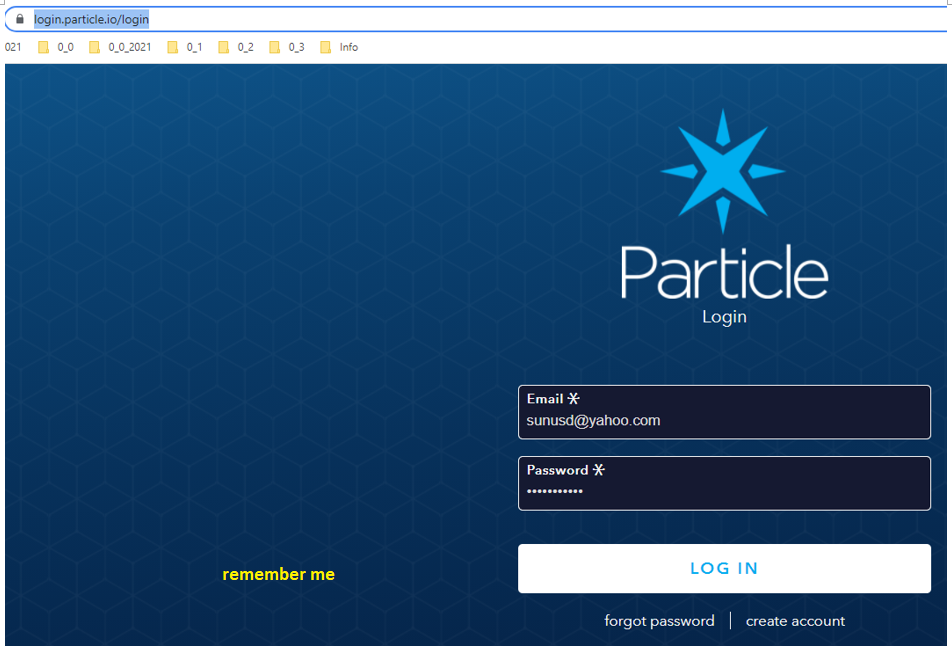


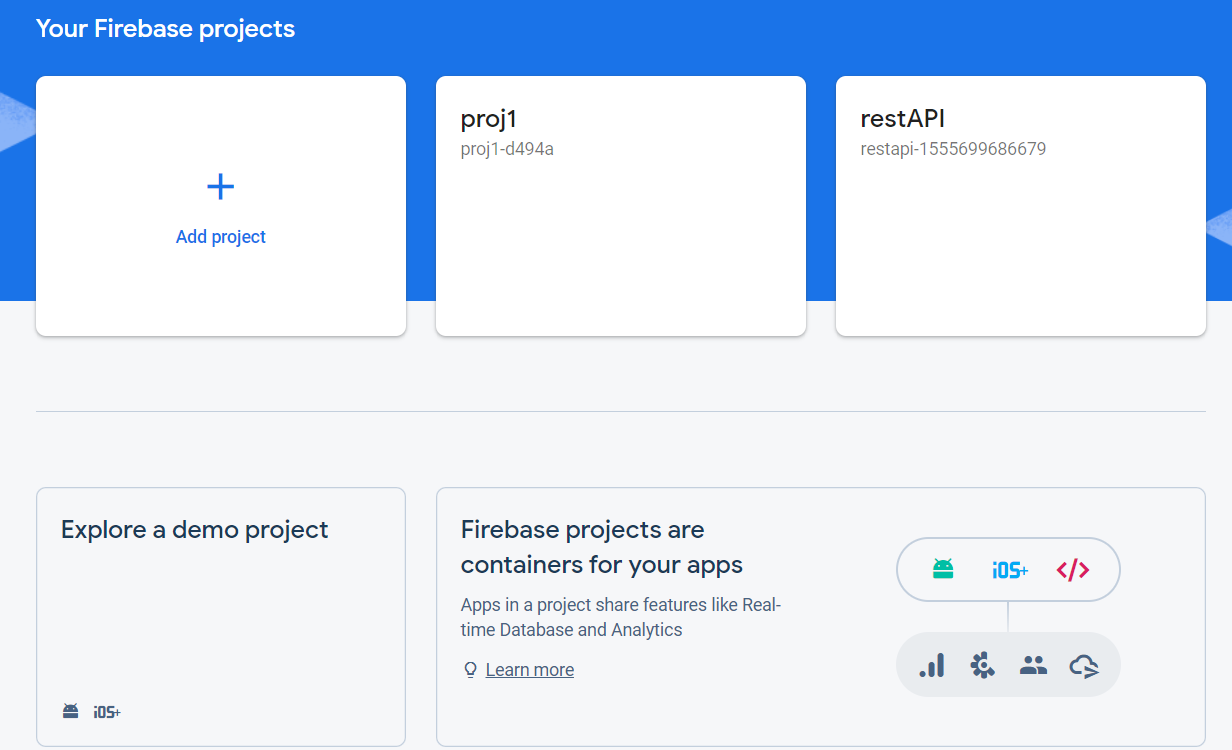
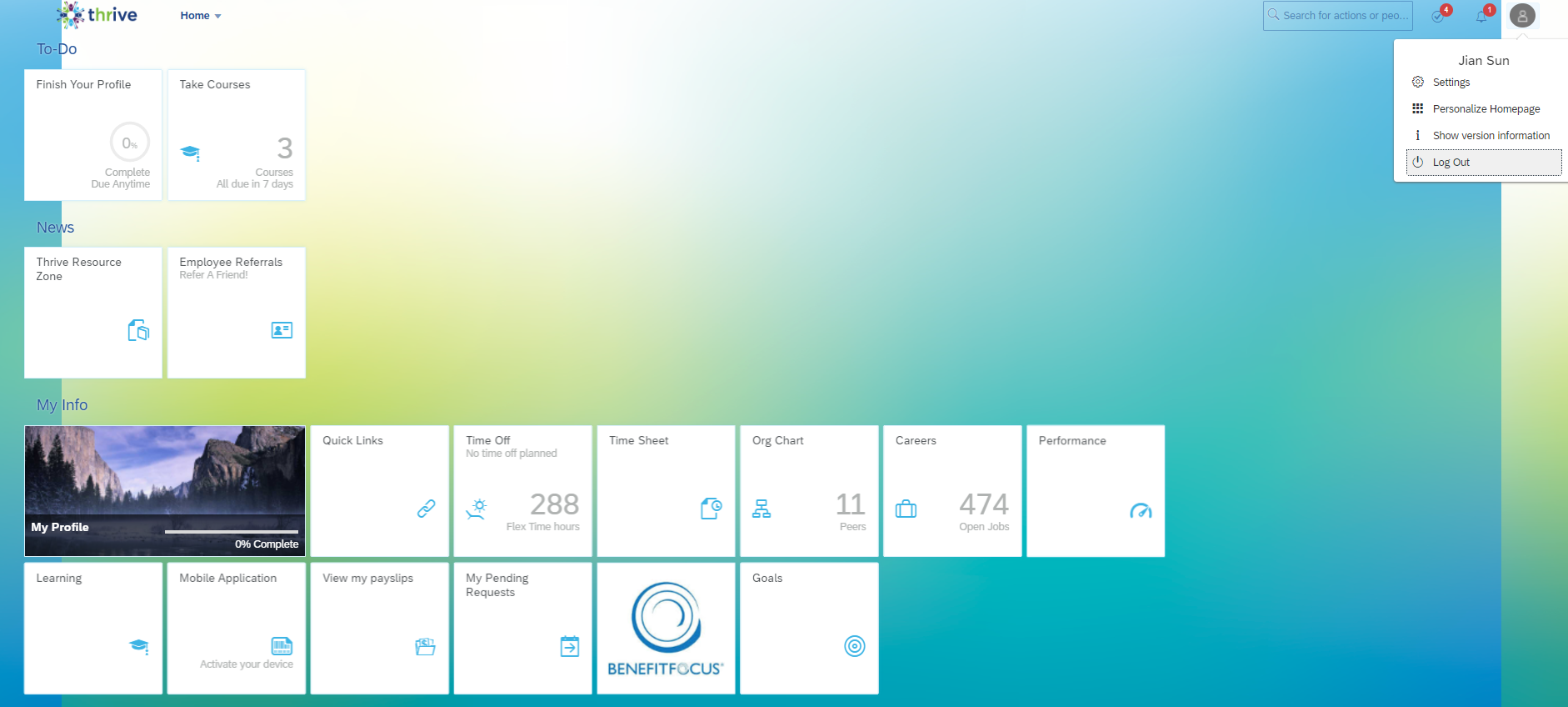
1. Group/workflow
2. Group/reportA
3. **UI**
4. Dashboard (log-in), web-browser based

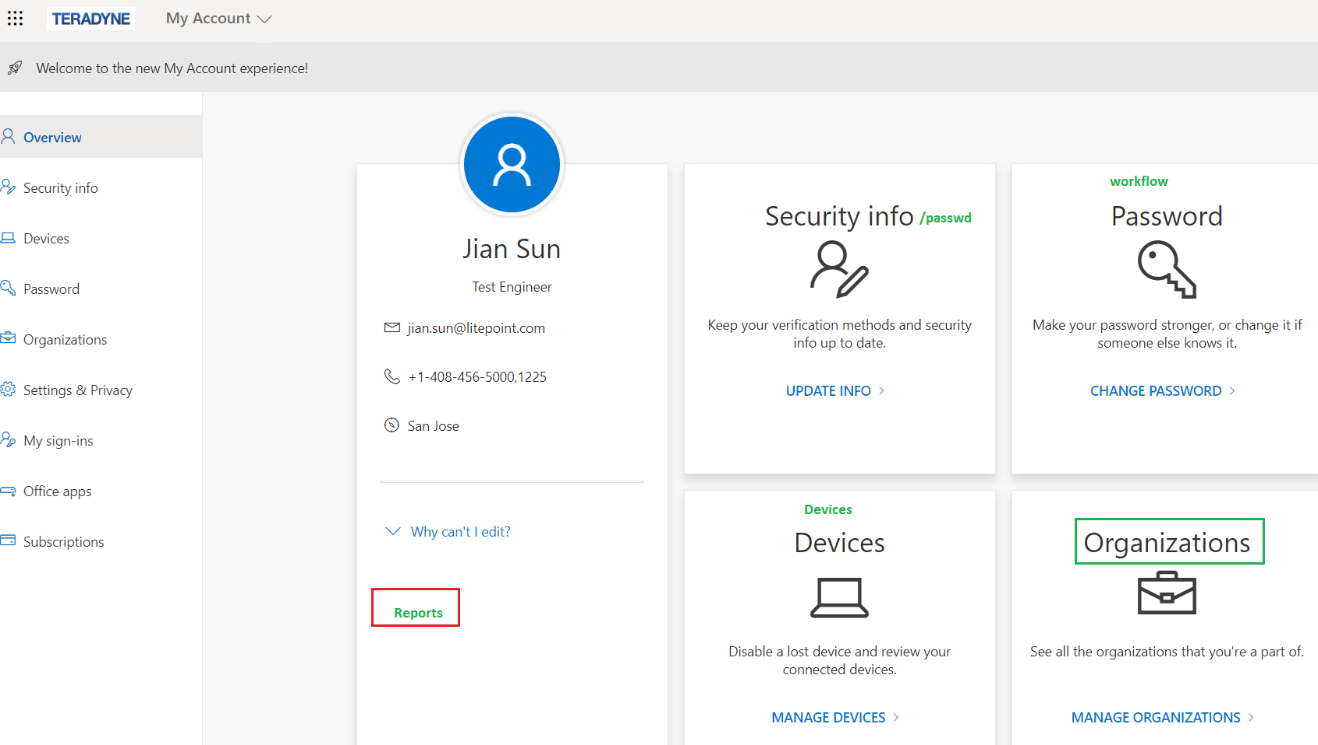
* <https://myaccount.microsoft.com/?ref=MeControl>
* <https://console.firebase.google.com/?pli=1>
* <https://hcm41.sapsf.com/sf/home?bplte_company=teradynein&_s.crb=gwOBD%252f5EHG6bPgf%252fq3Agz4u2O4nZq8IDKT6uE%252fpwxwM%253d#Shell-home>



<https://account.meraki.com/login/dashboard_login?go=%2F>







##### Administrators

There are two basic types of dashboard administrators: **Organization administrators** and **Network administrators**.

* **Organization administrators (Organization ID)** have complete access to their organization and all its networks. This type of account is equivalent to a root or domain admin, so it is important to carefully maintain who has this level of control.
  + **Organization - Read-only**: The user is able to access/view most aspects of network and organization-wide settings, but is unable to make any changes.
  + **Organization - Full**: The user has full administrative access to all networks and organization-wide settings. This is the highest level of access available.
* **Network(Group) administrators – Group ID** have access to individual networks and their devices. These users can have complete or limited control over their network configuration, but do not have access to organization-level information (licensing, device inventory, etc).
  + **Network - Guest ambassador**: The user is only able to see the list of Meraki authentication users, add users, update existing users, and authorize/de-authorize users on an SSID or Client VPN. Ambassadors can also remove wireless users, if they are an ambassador on all networks.
  + **Network - Monitor-only**: The user is only able to view a subset of the **Monitor** section in the dashboard and no changes can be made. This can be useful for proving networking monitoring access to customers in service provider deployments.
  + **Network - Read-only**: The user is able to access most aspects of a network, including the **Configure** section of the dashboard, but no changes can be made.
  + **Network - Full**: The user has access to view all aspects of a network and make any changes to it.

1. Tabs To be filled

* https://console.firebase.google.com/

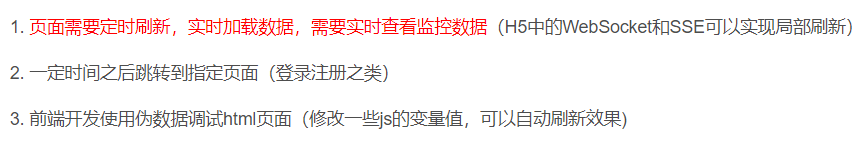
1. **Create a user account**

* Organization sys-admin issue user an organization ID and token. Token has scope that decides user rights.
* User use those two to create a user account with username/email and passwd

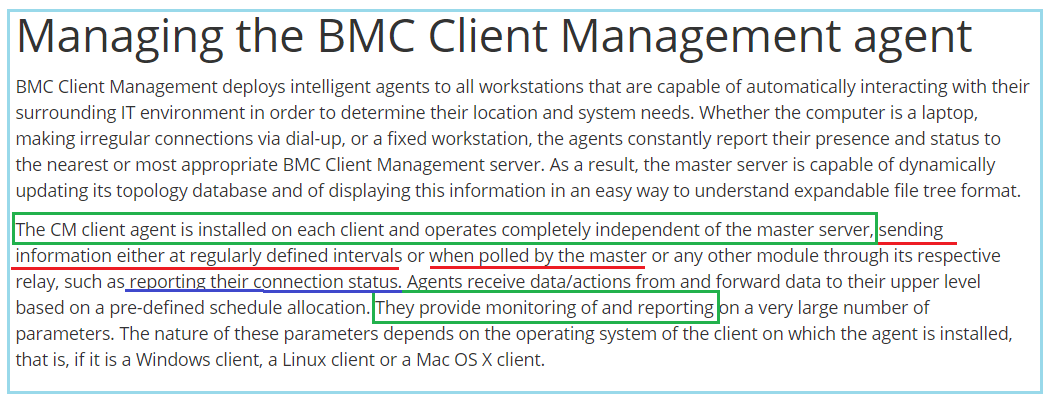
1. User info
2. Device/tester info, current status
3. Test history/summary
4. Managing tests

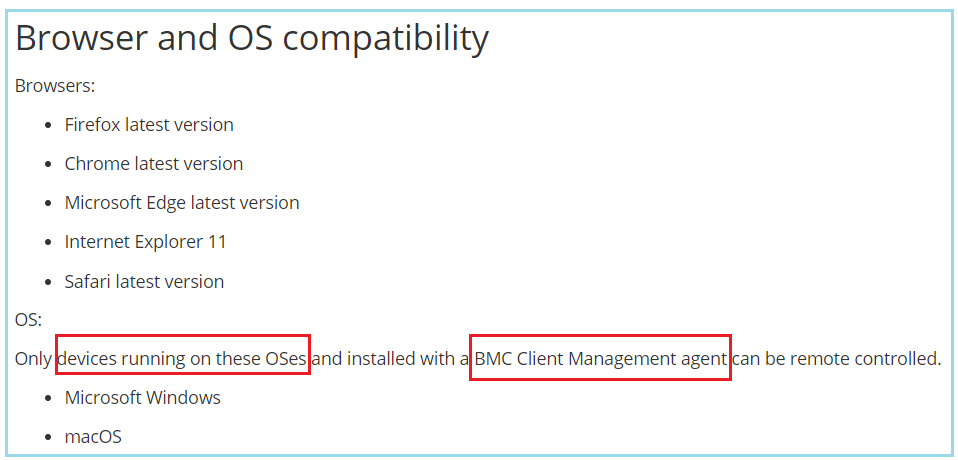
* Button-to start the test, which communicate with agent, agent then kick off the test program (iqfactrun\_console)
* Once test started, go to monitor page to see test progress and status
* Monitor webpage auto-refresh to show real-time test status.

1. Run tests in a console as desktop console: iqfactrun\_console in windows.
2. Tbd



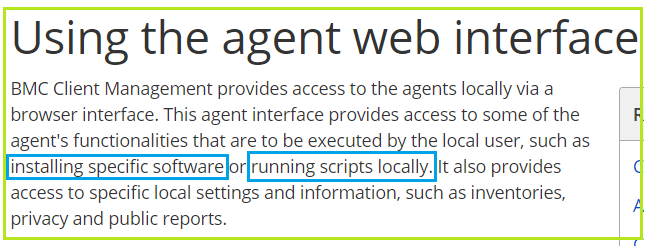
1. Agent (Litepoint Client Management agent)

****



1. Agent is only interface that server can use to know devices/testers and their activities.
2. I want agent to monitor and report/update test/device status to server; so user can access those info through UI; not through agent itself. Agent regularly poll the test program and get updated for pass/fail statistics, **test run time so far**; No other test results details. That’s when csv results uploaded and server run analytics.
3. Not decided that agent can send **real-time big logs** to server; so far comfortable with sending out real-time test summary, like failures, passes and # of tests run.
4. Agent needs to be able to communicate with IQfactrun\_console to collect test summary and maintain communication with server.
5. Agent can have a GUI, that user can access locally. Configure agent through GUI, to set up devices, testers that agent need to monitor.

* Set up IQrun\_console through agent, pass tests parameters like #runs, test flows.
* Start tests locally.
* Or UI can define those parameters, send those command through server; server can send those command to agent. That’s how users can run Iqfactrun\_console through UI.

 https://docs.bmc.com/docs/bcm126/en/using-the-agent-web-interface-738019744.html

1. kkk

<https://docs.bmc.com/docs/bcm126/en/managing-the-bmc-client-management-agent-738019718.html>

1. IQfactrun\_console Console based testing (how to remote running it)

**REST API 🡪 browser🡪regular IQfact+ automation**

**All the information is**

How does server control Device through Rest API:

1. Read Device info:

* Device info is stored in database.
* Test flow in txt/json file, which can be converted to json files(test flow).

1. Control device

* Add/update Device a flow
* Device to run a flow through API or UI (there is a run button)

1. API to run a test: Create a new test

POST ../v1/device/test\_name

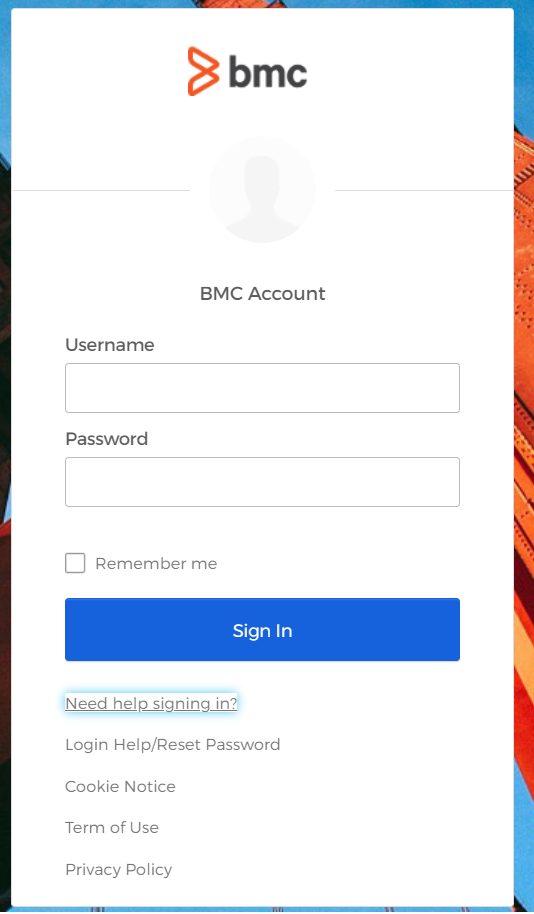
Dev name/id/ip, test\_flow\_name, test\_name

1. Once server receive the request, it will update status of device and pass the demand to agent; agent will initiate test program with all the info passed by server.

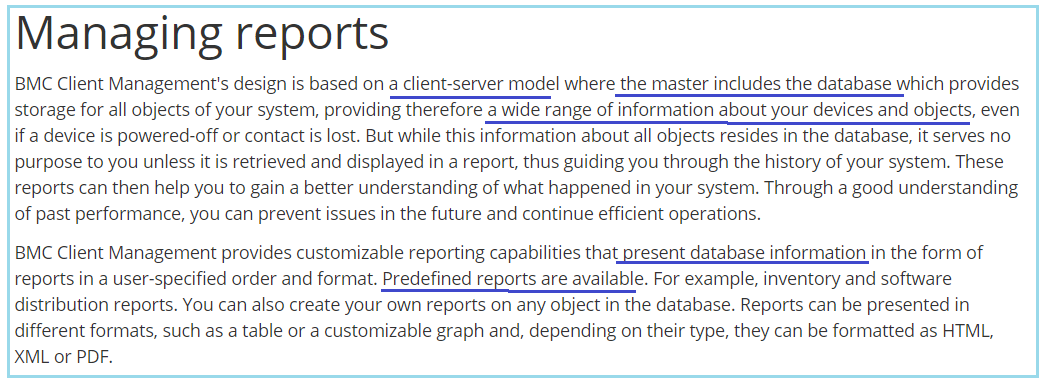
* Question? How the test program gets work-flow. So agent should have a functionality that can trigger server to download workflow to device PC. Or test flow can be retrieved from local PC; API needs to pass the “path to the workflow”.

1. **API**
2. **/template/workflow**
3. Template for create/modify workflows
4. API for analytics (like iqramp report run by server, user just need to upload data csv/Jason files) ; result csv/Jason files need to be uploaded first to the server, server will parse the log and save testcase/results to database so that API can retrieve during analytics.
5. Log in

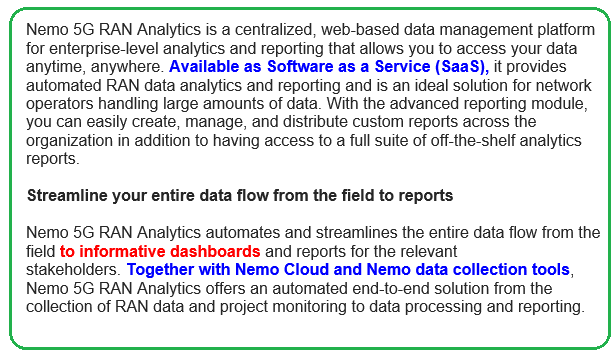
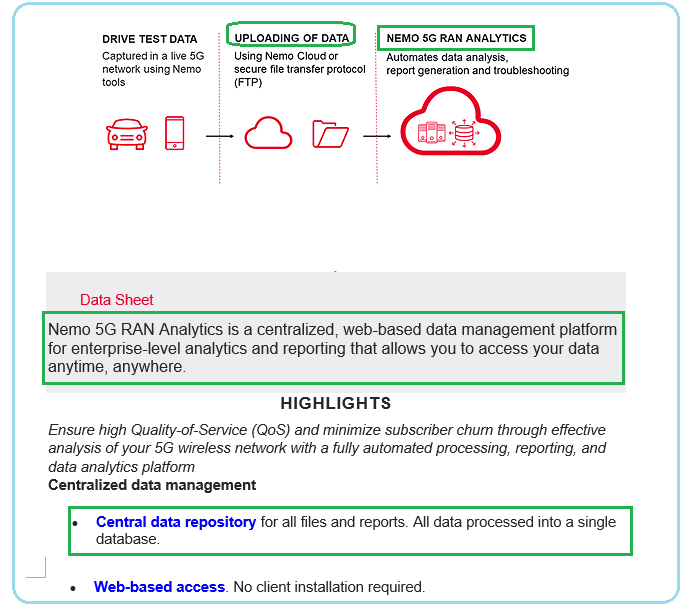
<https://bmc.okta.com/login/login.htm?fromURI=%2Fapp%2Fbmc_docsbmccomnew_1%2Fexk1c4kk7j57sck0T1d8%2Fsso%2Fsaml%3FSAMLRequest%3DfZJPT4NAEMW%252FCtl7WXZbpG5Kk9oebFKVCHrw0izLaBHYRWax9dsL1D81MT1tMjvvN%252FNeZoayKmuxaO1O38NbC2idQ1VqFMNHSNpGCyMxR6FlBSisEvHiZiO464m6MdYoUxJngQiNzY1eGo1tBU0MzXuu4OF%252BE5KdtTUKSjOj0E0r5SpT0XiXp6kpwe5cREN7JqfRXZwQZ9UtkWvZ437Fvc4UVg5iWdd9YdsTu7cradhvGYVDwdSkKIJXP0BVeAnLprTH92aIs16FZMshmyrvMgv8Z6WCFLwgnfjsgo2nKfPHHnRtiC2sNVqpbUi4x%252FnI4yPOEm8iOBNj%252F4k40Zf1q1xnuX45n1N6bEJxnSTR6GjyERocDHYNZD7rFxTD4OYk%252F%252FNY%252BR06mf8bMf5EPKMn%252FOOwWtx2wPUqMmWuPpxFWZr9sgFpISSM0PlR8vcy5p8%253D%26RelayState%3Dss%253Amem%253A8c2f95f04ca2c3d857769018fc024f40eaaab8488137daec48899688e1f239ef>



1. Reporting

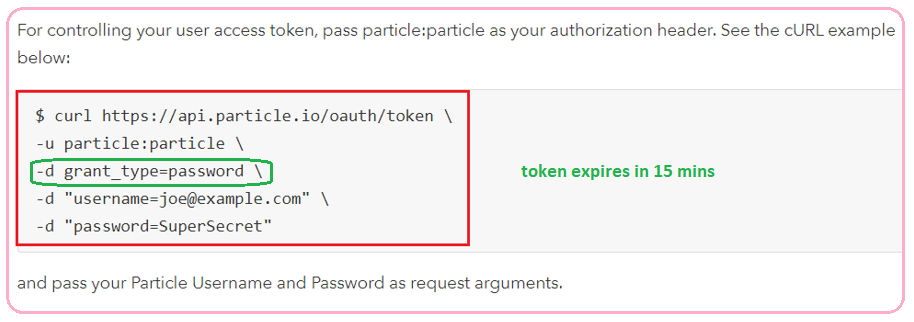


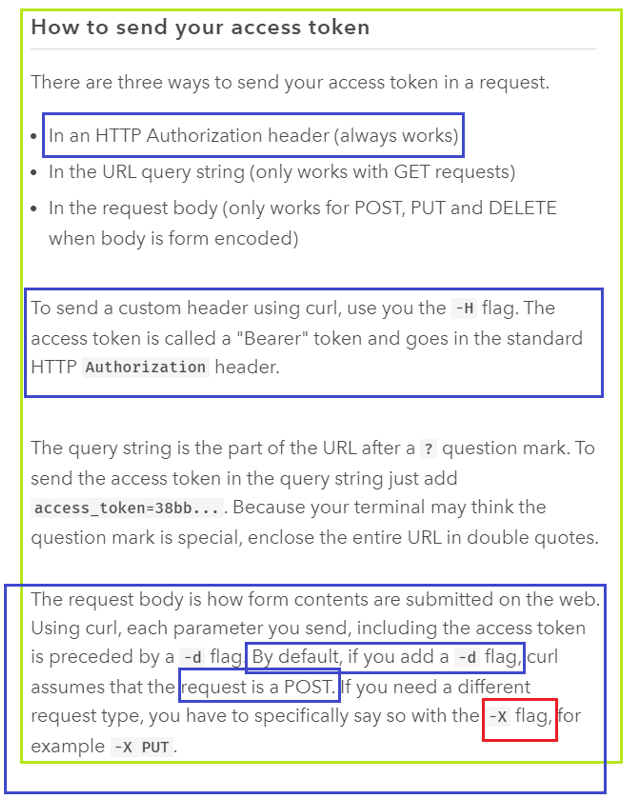
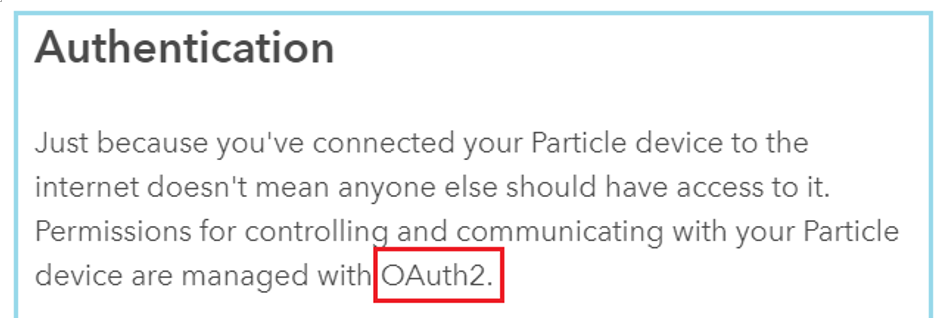
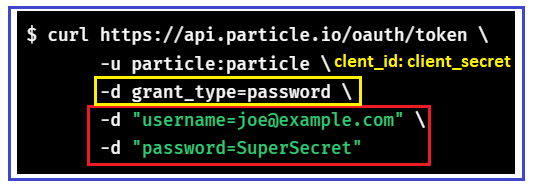
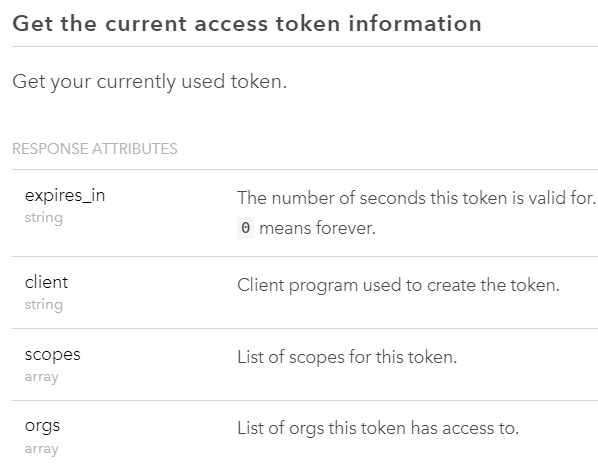
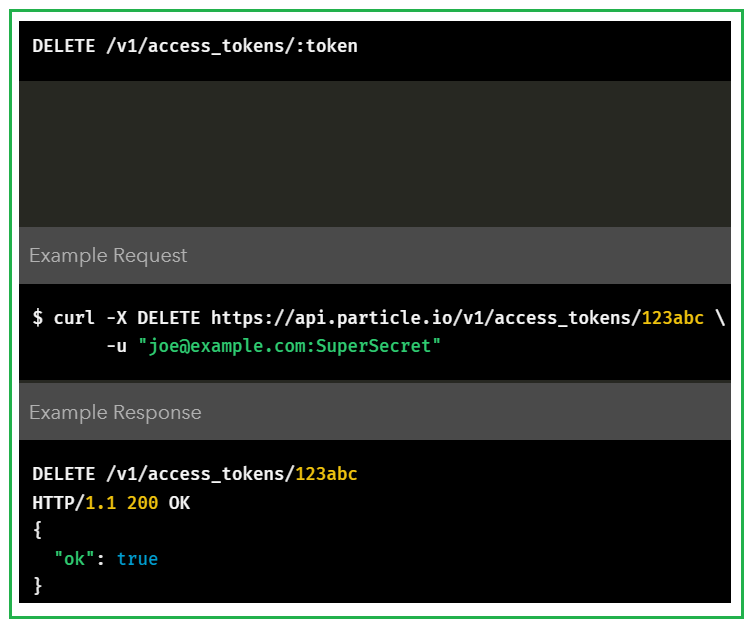
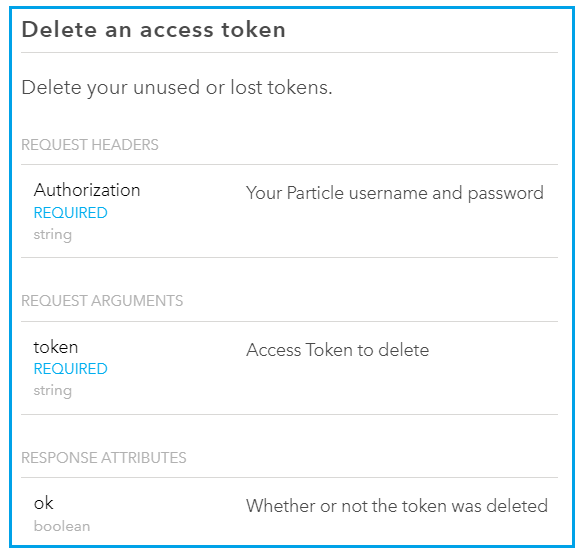
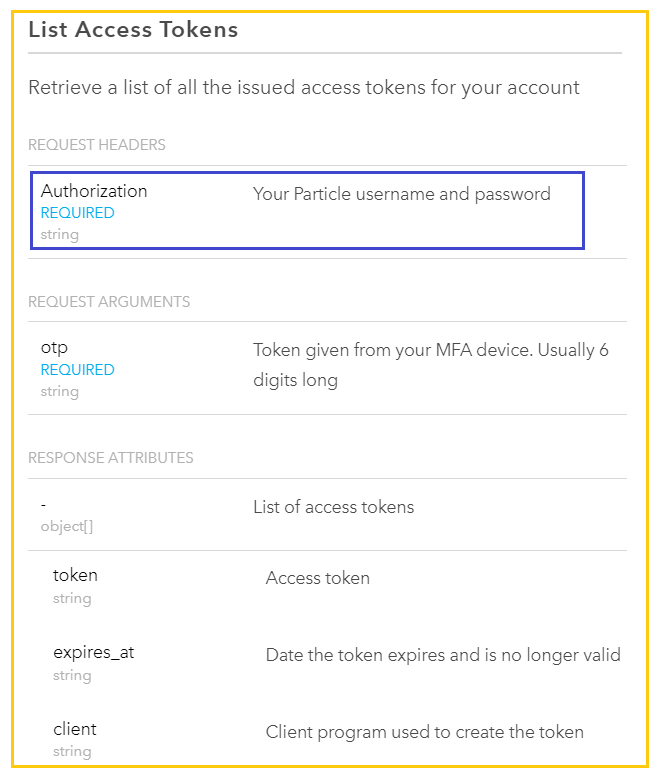
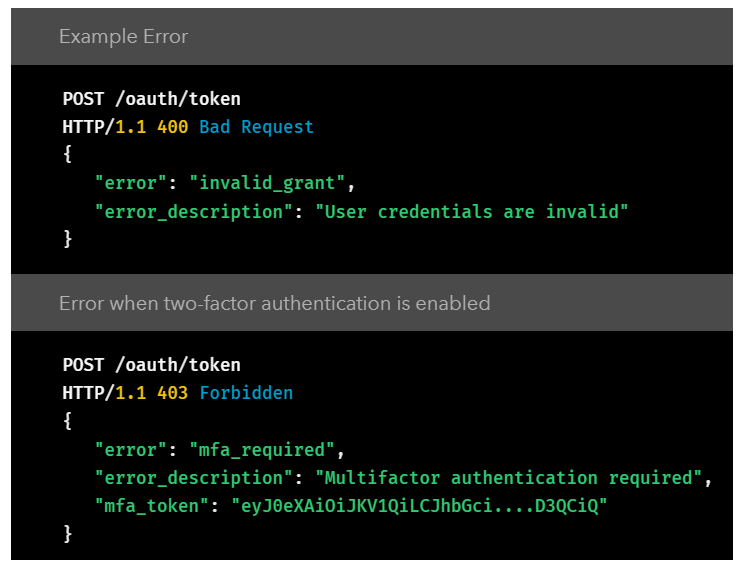
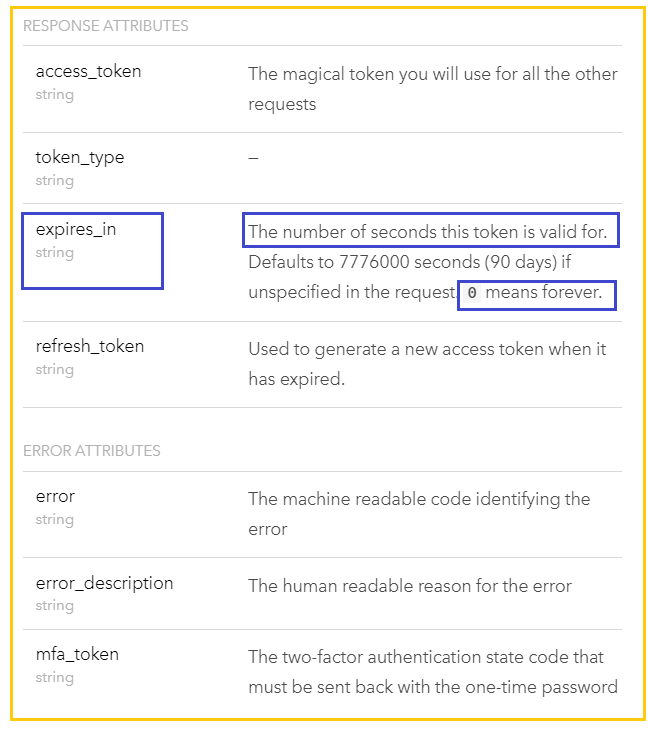
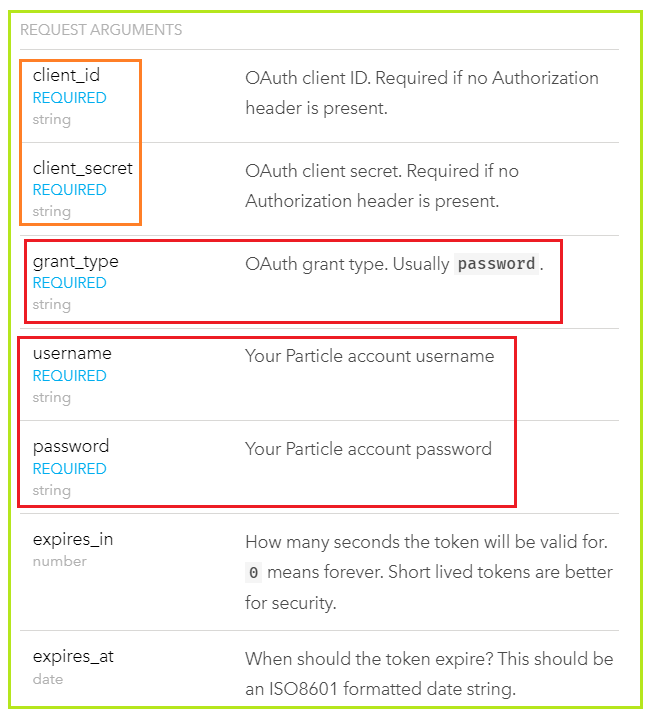
1. General report about DUT history
2. Report about test summary about a particular test, test\_name can be looked up from UI
3. Many deployments will find that they benefit from some type of device reporting, or may have some kind of mechanism in place for **monitoring device status**. Options for monitoring devices include standard dashboard monitoring, and API device status reporting.
4. **Upload csv or json files**. Generate customized IQramp report for certain test cases through API with parameters; report is run by server analytics. Can be downloaded to client through UI or API.

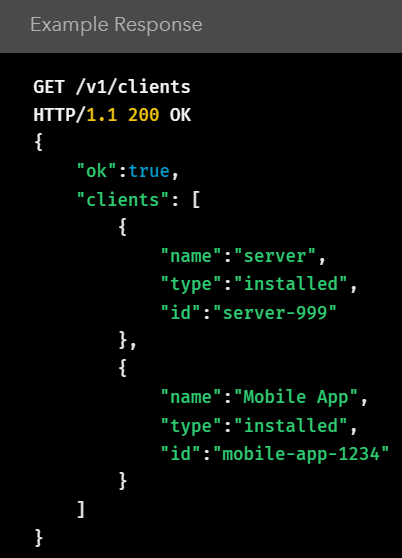
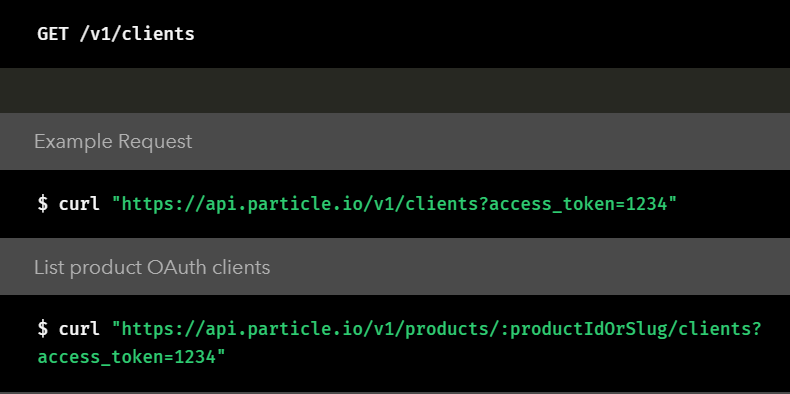
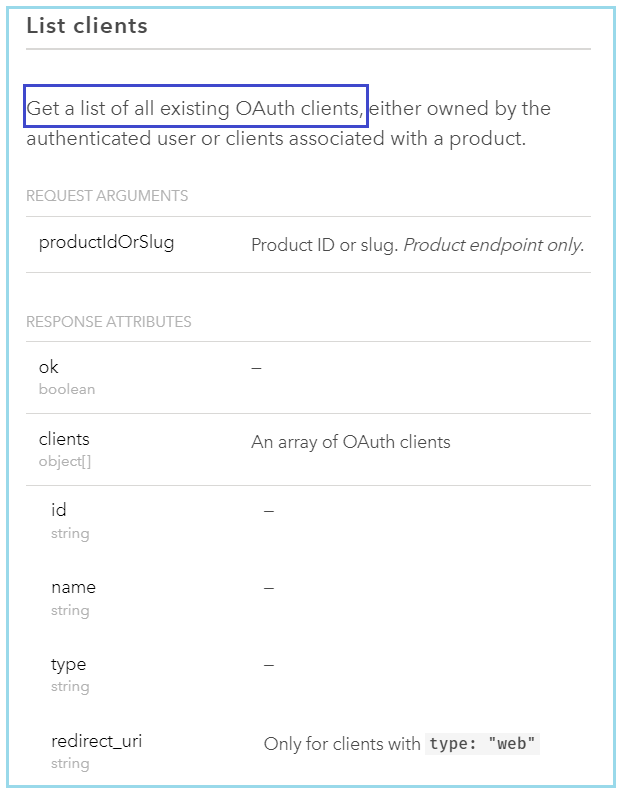
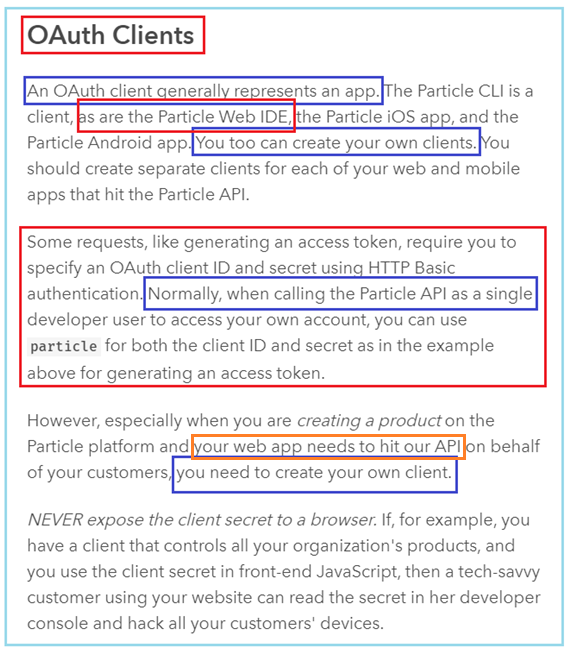
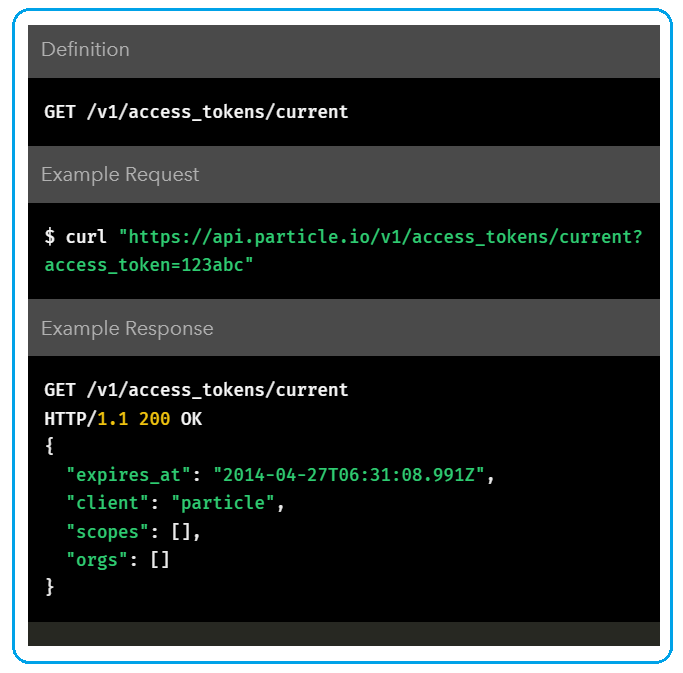


1. Fff
2. API access token or API key
3. How to generate and update token or key

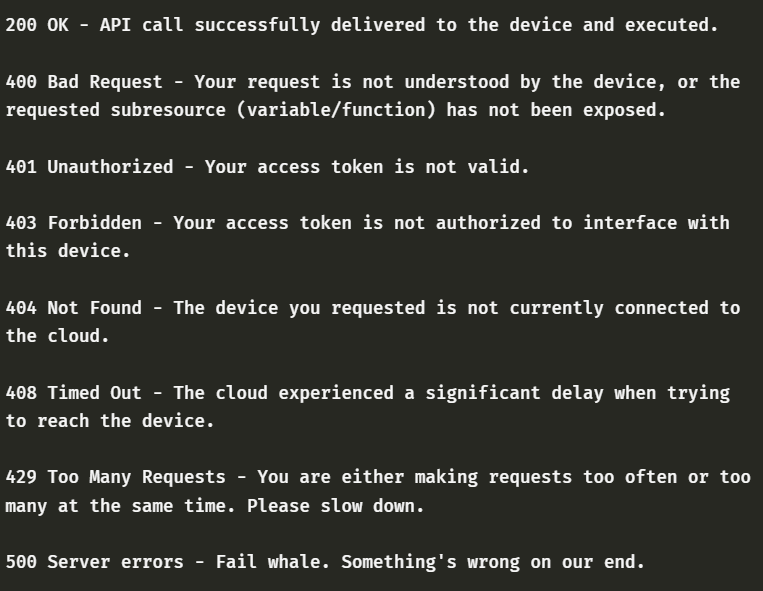
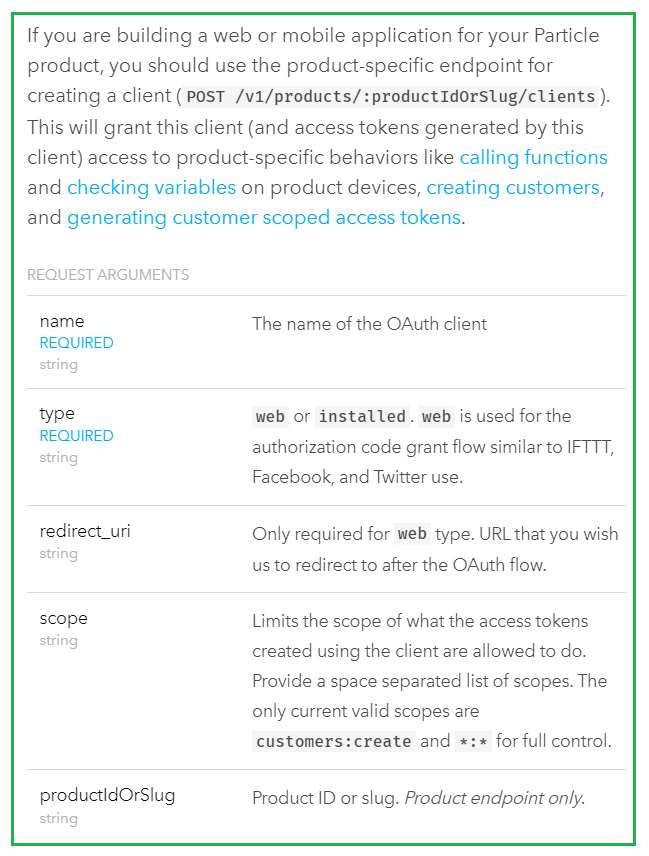
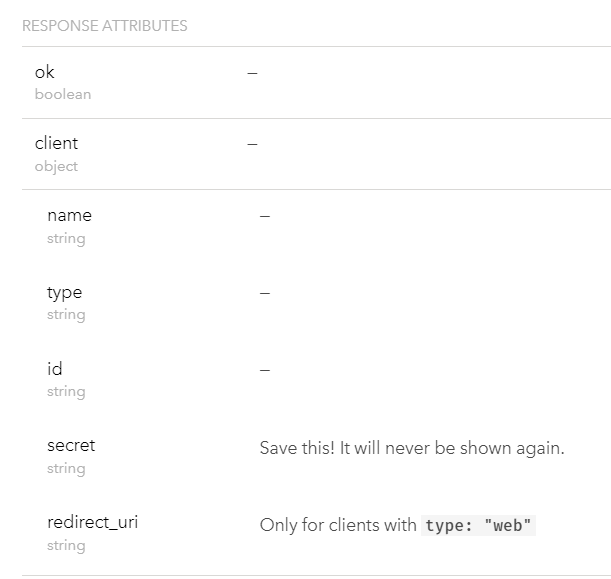
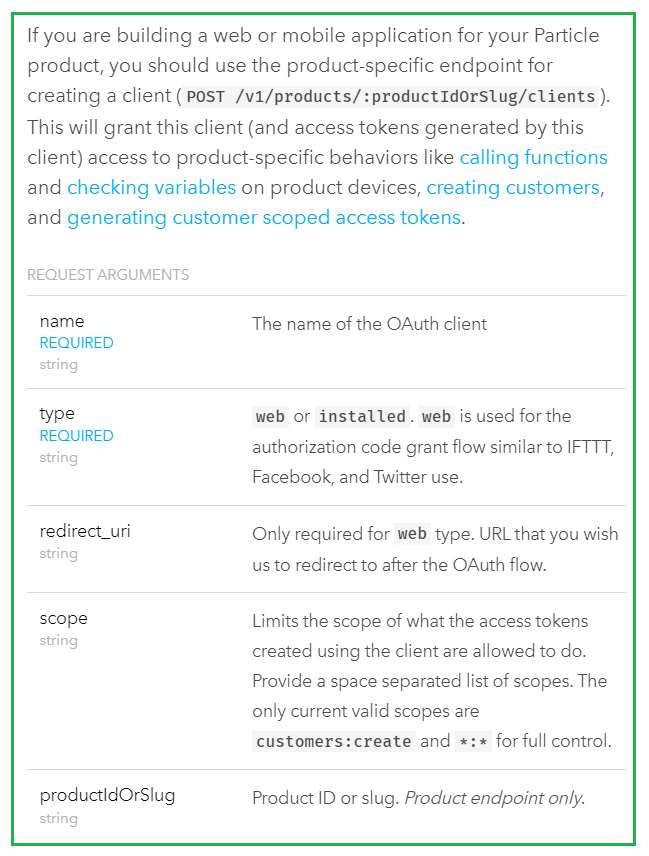
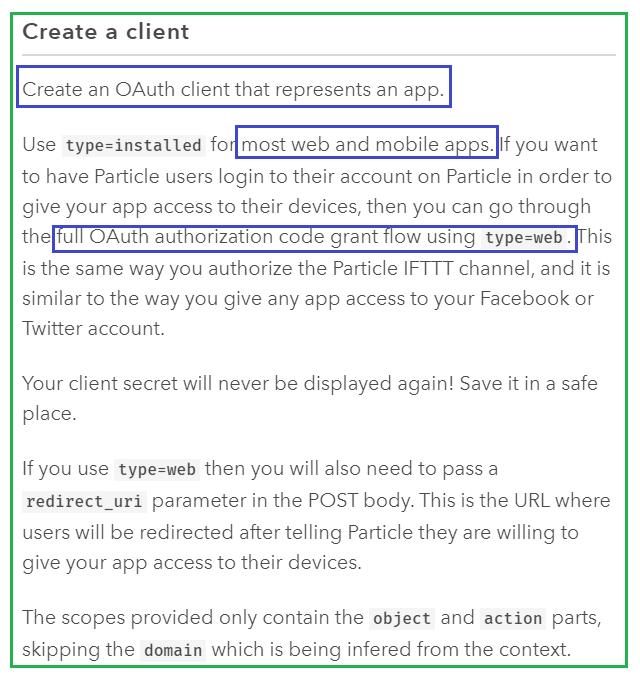
* Get a token through UI
* UI will also show current token being used, and expired token
* Got a token through API call



********

Scope can be defined as permission for access/modification for test\_flow. ****

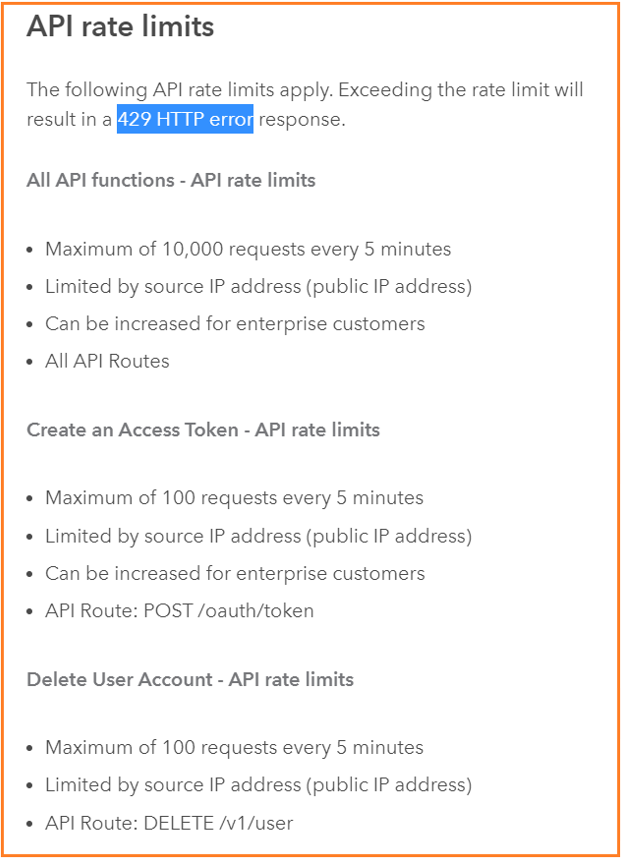
**Create a client for a app, and the client can request a access token for the app.**

****

1. How to use and when to use

* Each user manage devices and workflows
* Each user can have upto 5 access tokens with different scope, which allows: **personal token and token to access users from group (read/only)**
* Create/modify/delete devices
* Create/modify/delete workflows
* Upload/download test summary/logs.
* Allows to manage tests: running, stopping a test.
* Access token scope
* Get, list , update, remove, 🡪device
* Get,list, update, delete -> work flows
* get, list, update, delete -> analysis
* run/stop tests.
* IQfact\_console test SDK update/download

1. **API rate limits**



1. Need to decide token expiration time and etc.

* **15 days expiration**

1. Token specs for users and automation purpose.
2. Manage workflows
3. Manage device info
4. Each device sits on a host PC,
5. OS: windows, Linus and Mac
6. There is an agent on the host PC
7. Device ip is same as host PC ip. Device has id,name to identify itself.
8. **To install** on PC:

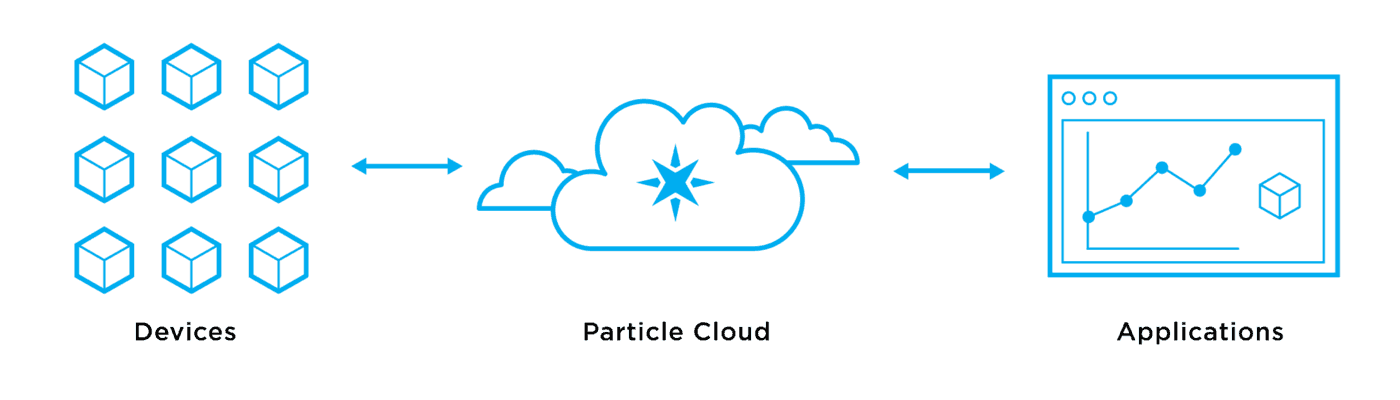
* Test engine: To run the test
* SDK/ agent: communicate with LP server
* Plus configuration of test-engine and agent

**Particle Cloud**

The defining feature of an IoT device is its ability to **send and receive information through the Internet**. This allows an IoT device to interact with other apps or with other IoT devices.

The Particle **firmware** (the operating system) on your Photon device has **built-in functions to allow your device to send and receive data through the Particle Cloud service**. This will allow your device to interact with web apps that you create.

Particle also has a JavaScript library that provides functions that will allow web apps to send and receive data through Particle Cloud. You can use these functions in your web app's JavaScript code, so that your web app can interact with your Photon device.



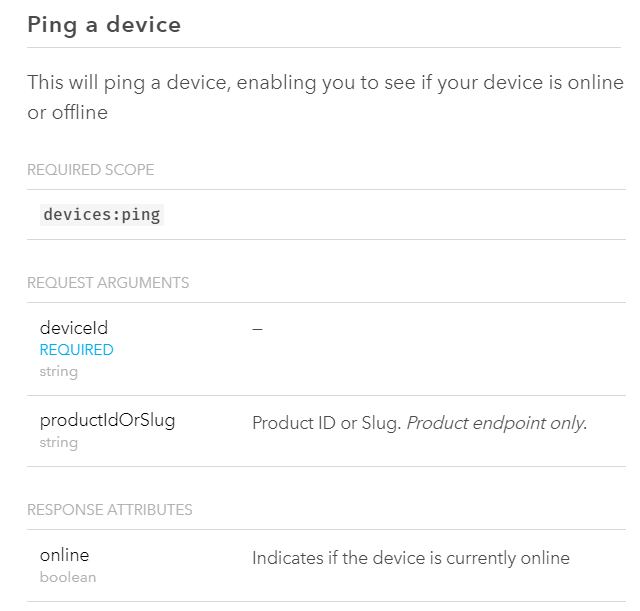
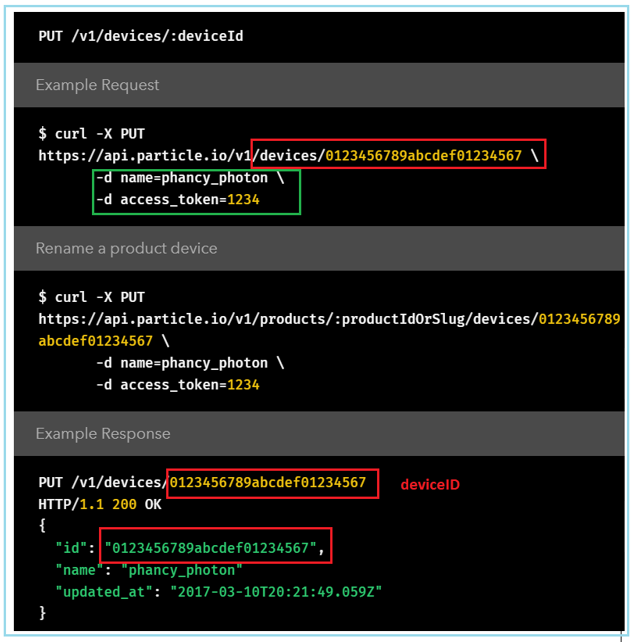
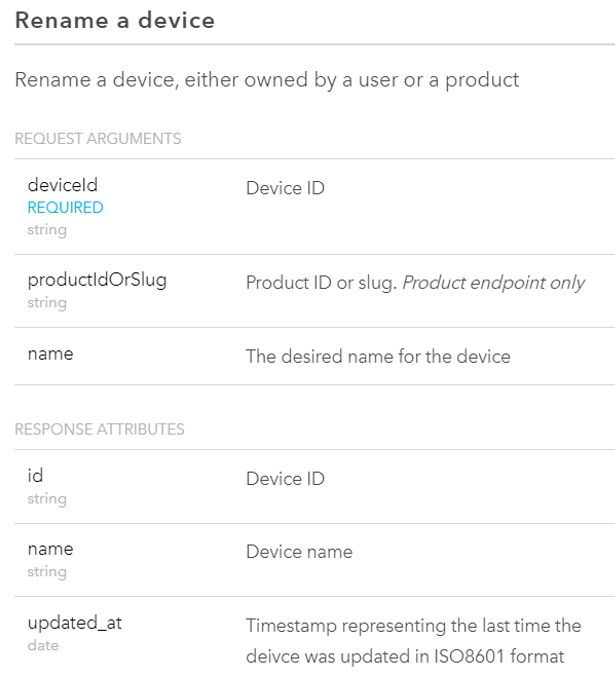
Particle Cloud

The three main ways your Photon device can interact with your web app through Particle Cloud are by:

* ​[sharing Photon variable](https://docs.idew.org/internet-of-things-project/references-for-wiring-and-coding/particle-cloud#share-photon-variable-through-particle-cloud): web app can read value of variable in Photon app
* ​[sharing Photon function](https://docs.idew.org/internet-of-things-project/references-for-wiring-and-coding/particle-cloud#share-photon-function-through-particle-cloud): web app can send commands and data to Photon app
* ​[sharing Photon event](https://docs.idew.org/internet-of-things-project/references-for-wiring-and-coding/particle-cloud#share-photon-event-through-particle-cloud): web app can receive event notifications and data from Photon app

You can also [combine these Particle Cloud interactions](https://docs.idew.org/internet-of-things-project/references-for-wiring-and-coding/particle-cloud#combine-particle-cloud-interactions) in more complex ways. For example, your Photon app could send an event notification to your web app, which then triggers the web app to read the value of a variable in the Photon app.

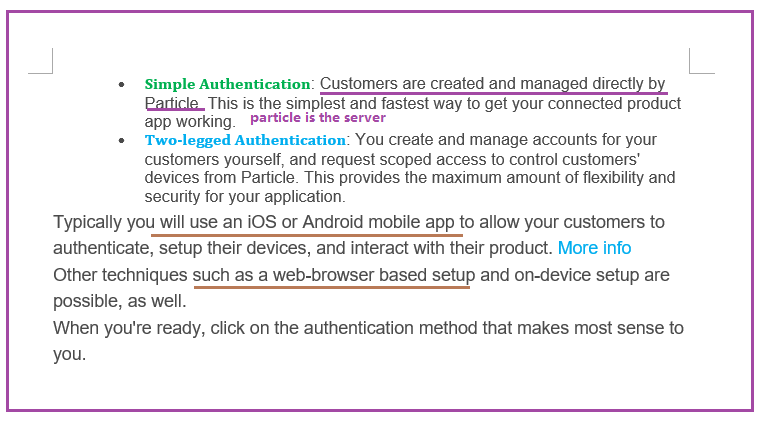
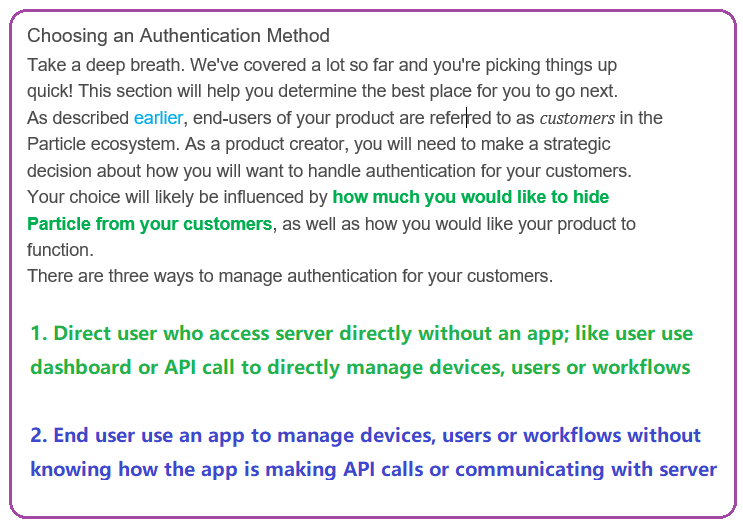
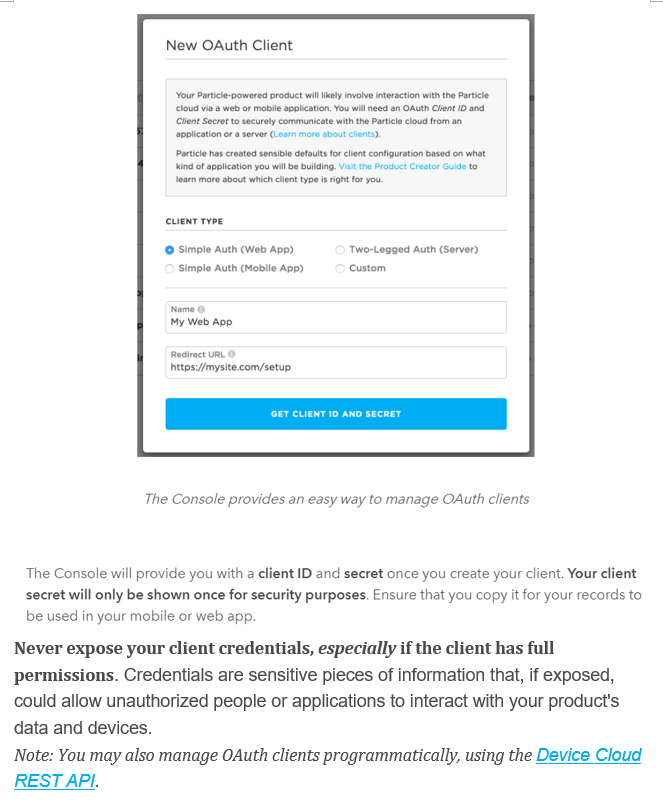
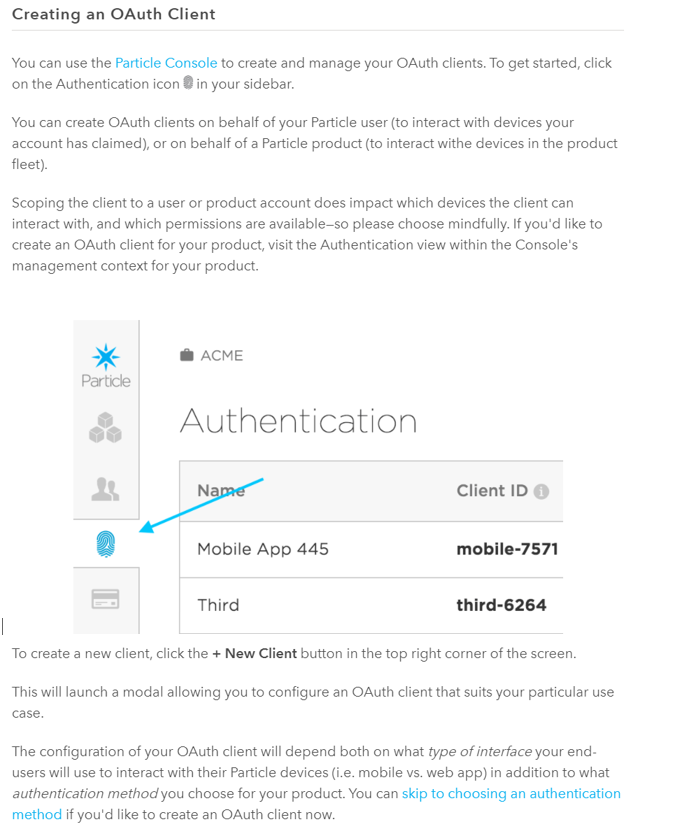
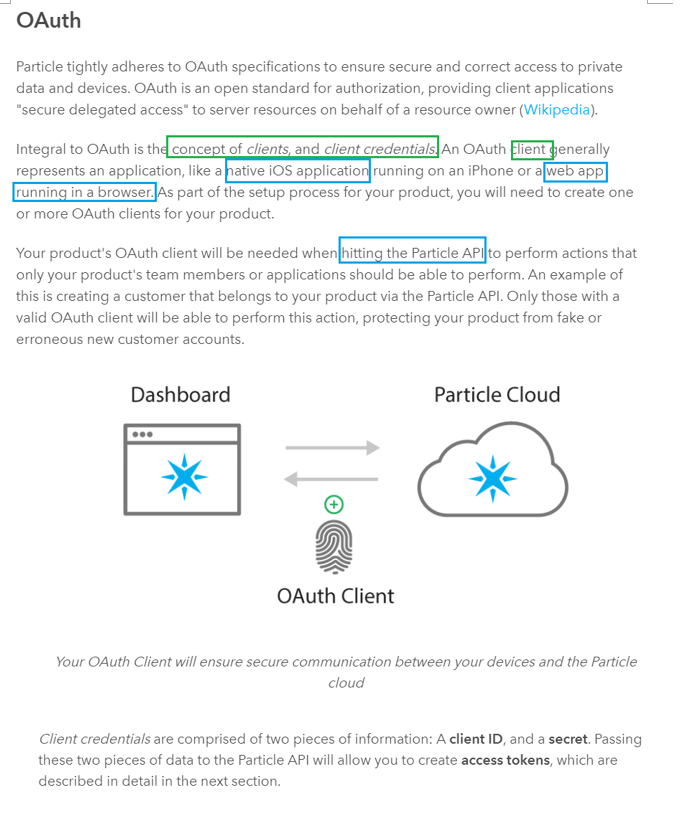
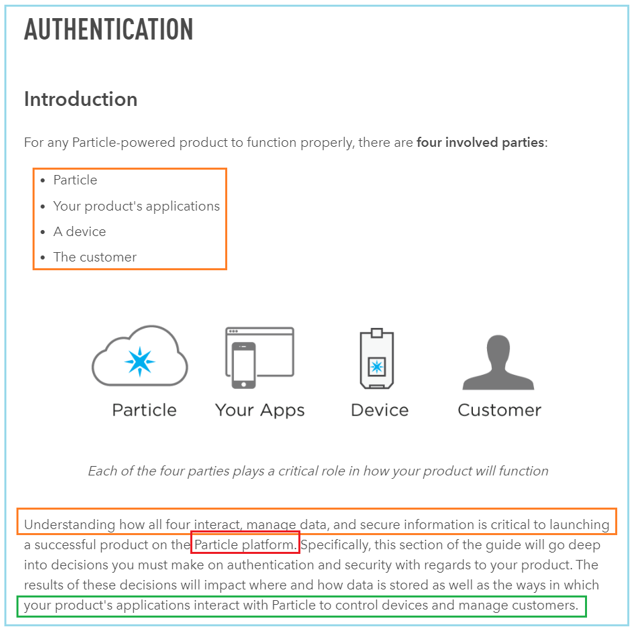
This reference contains code examples for both your Photon app and your web app. Many of the code examples will need to be modified, in order to be used in your apps. For example, you may have to change the names of variables, add custom code, etc.

****

1. Manage running tests.
2. Manage analytics
3. Authentication
4. Every employee user to apply user account

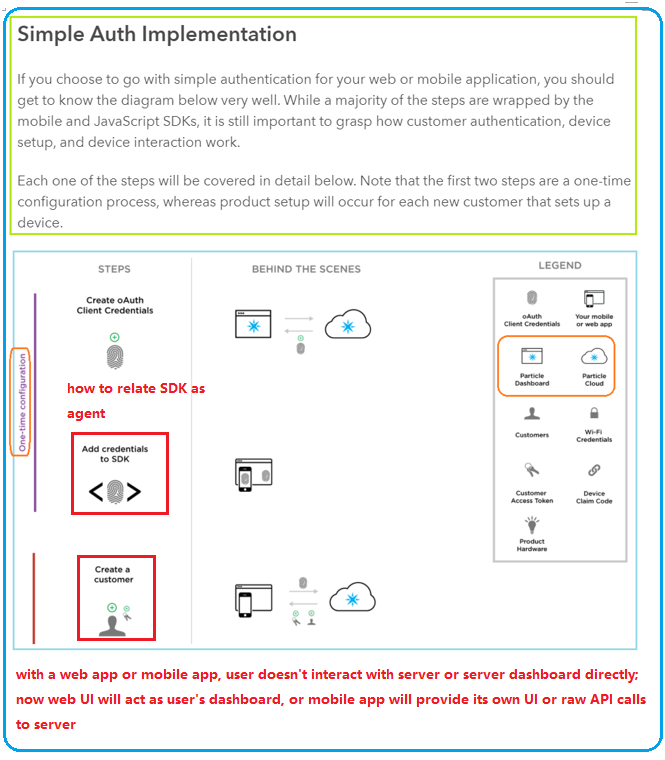
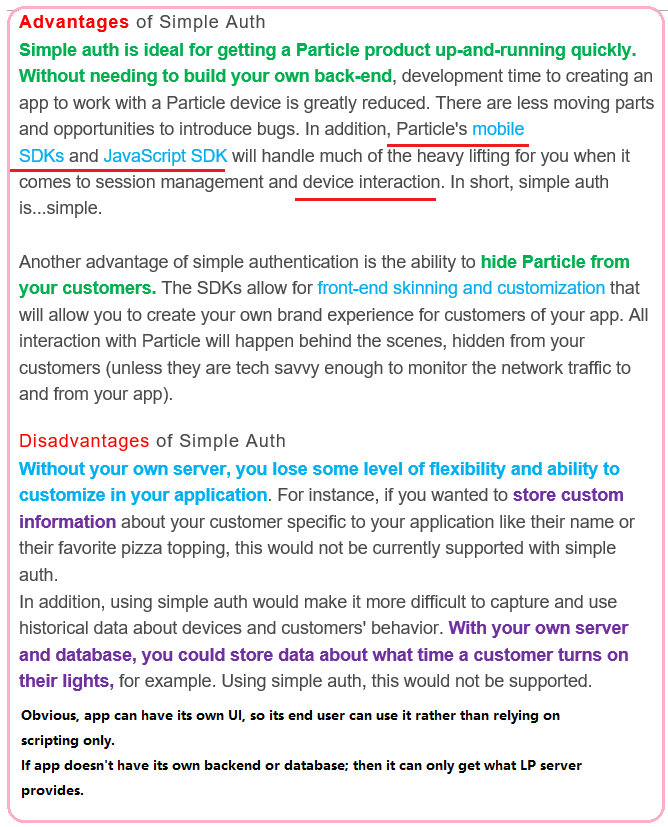
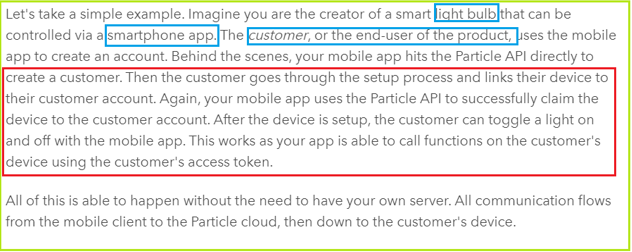
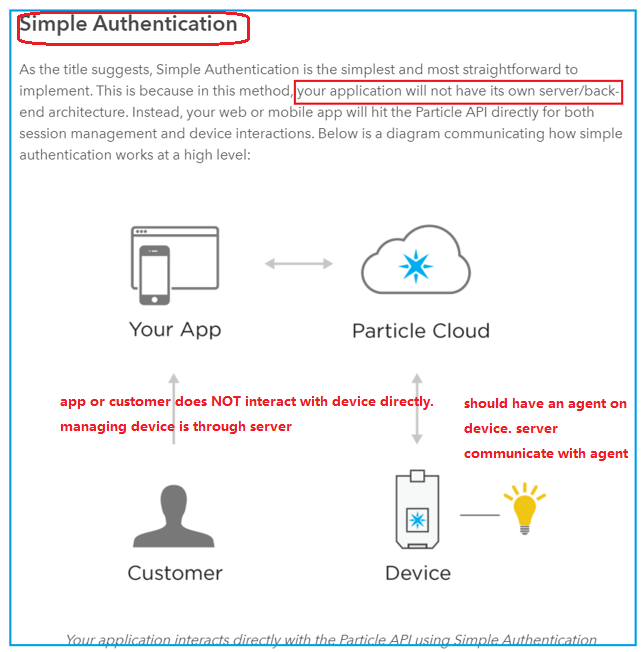
* Need company id to identify the organization
* Then followed by regular email/passwd,
* Create username, name, and other user information.

1. Client id and secret is issued from litepoint server.
2. User or its organization can apply for it , and need it for requesting access token in the future.



1. Simple authentication

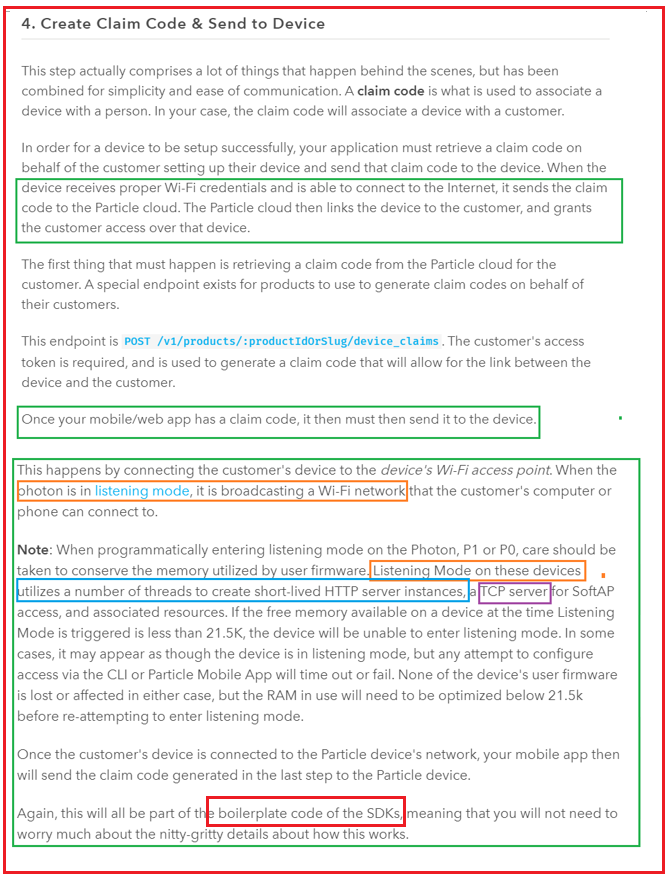
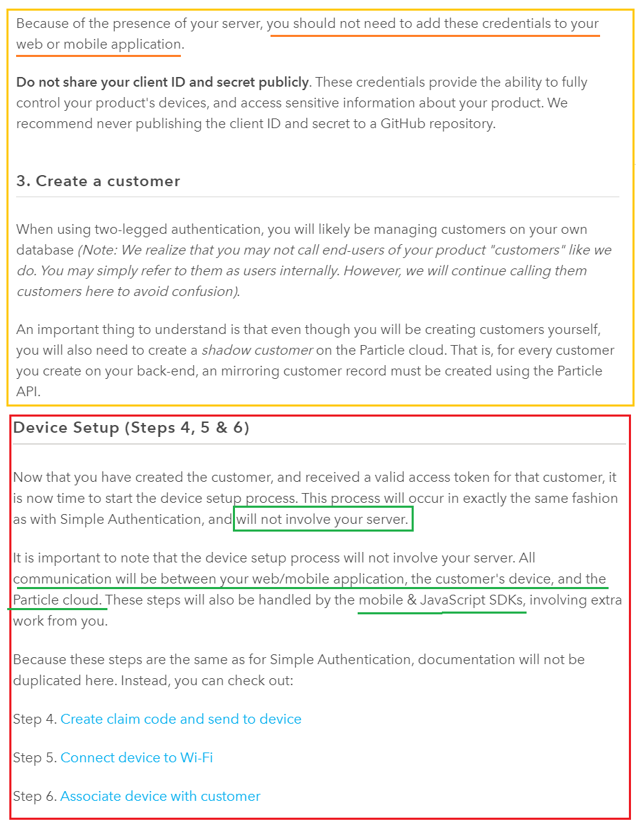
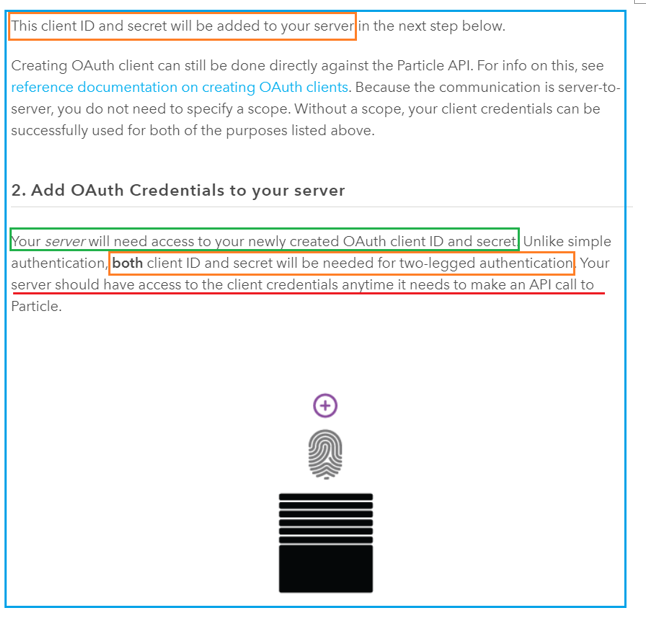
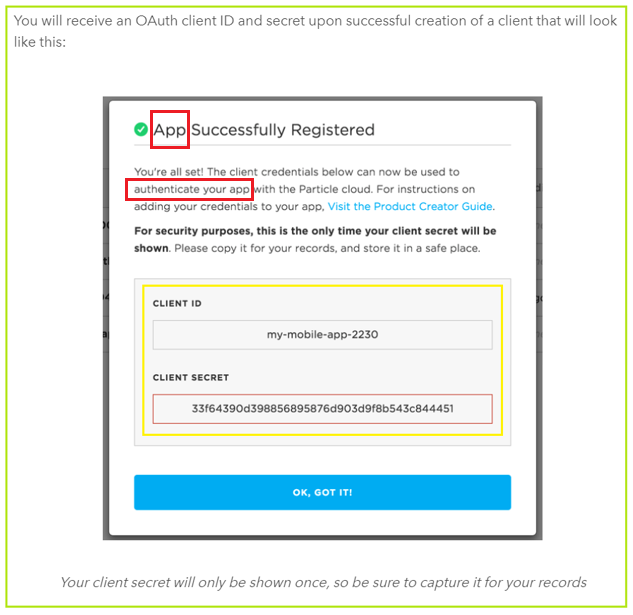
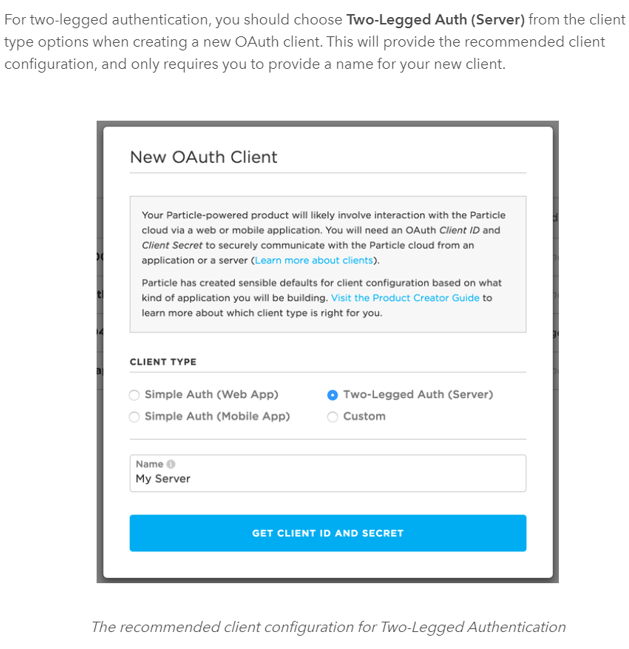
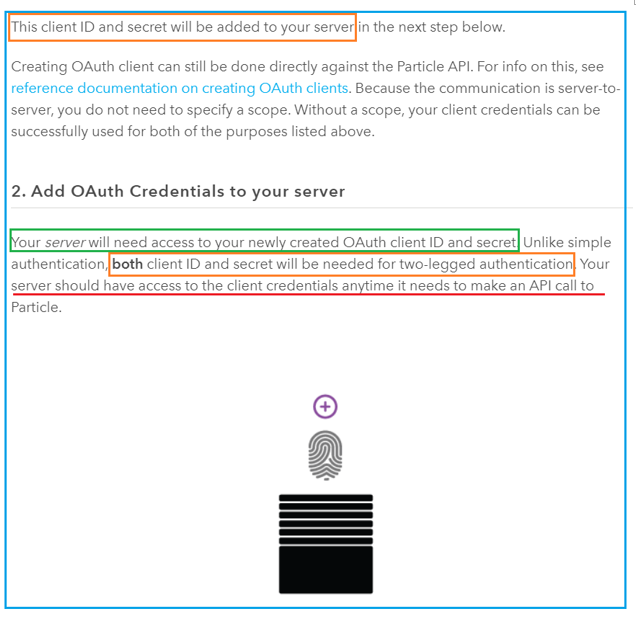
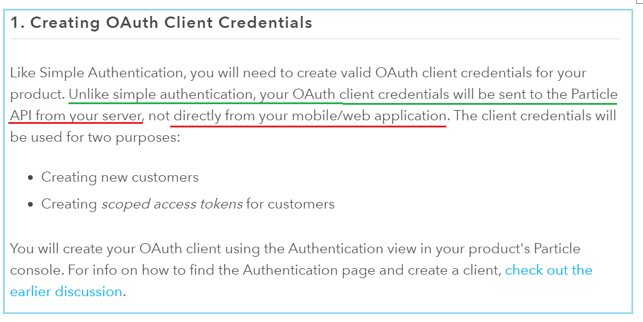
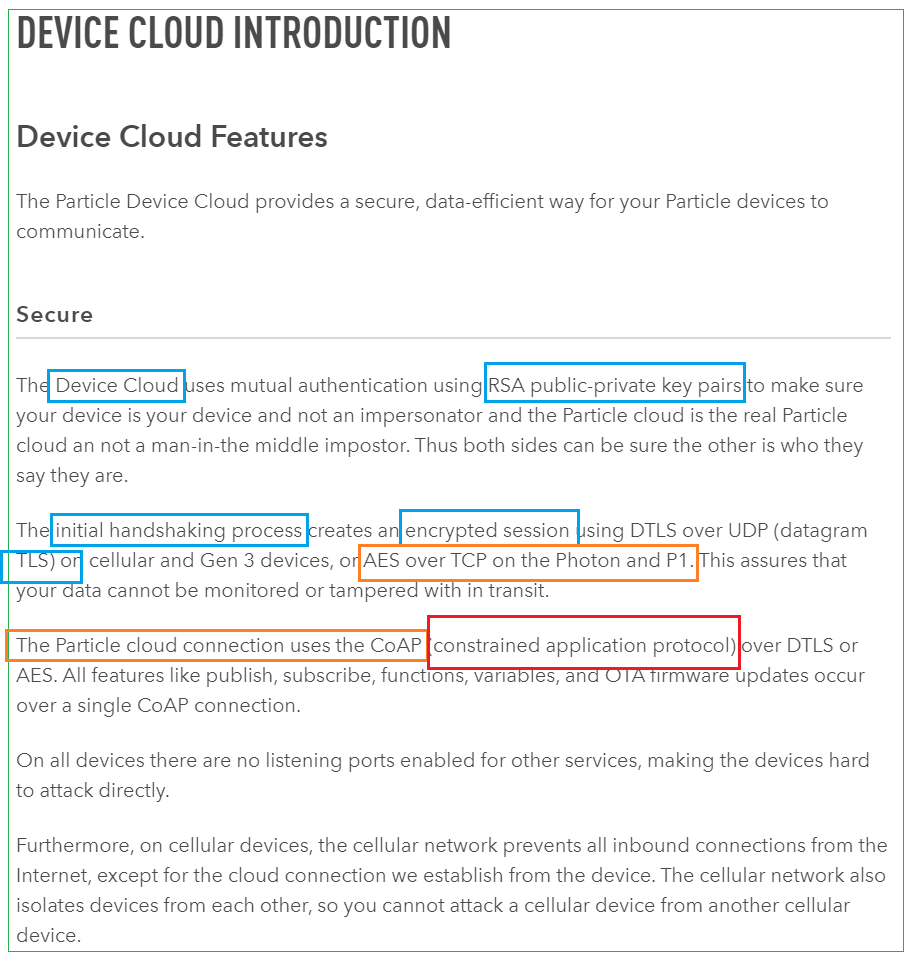
* App or automation platform does NOT have backend/server architecture. They make API calls directly to Lite server. Here app acts like automation scripts with UI for user friendly.

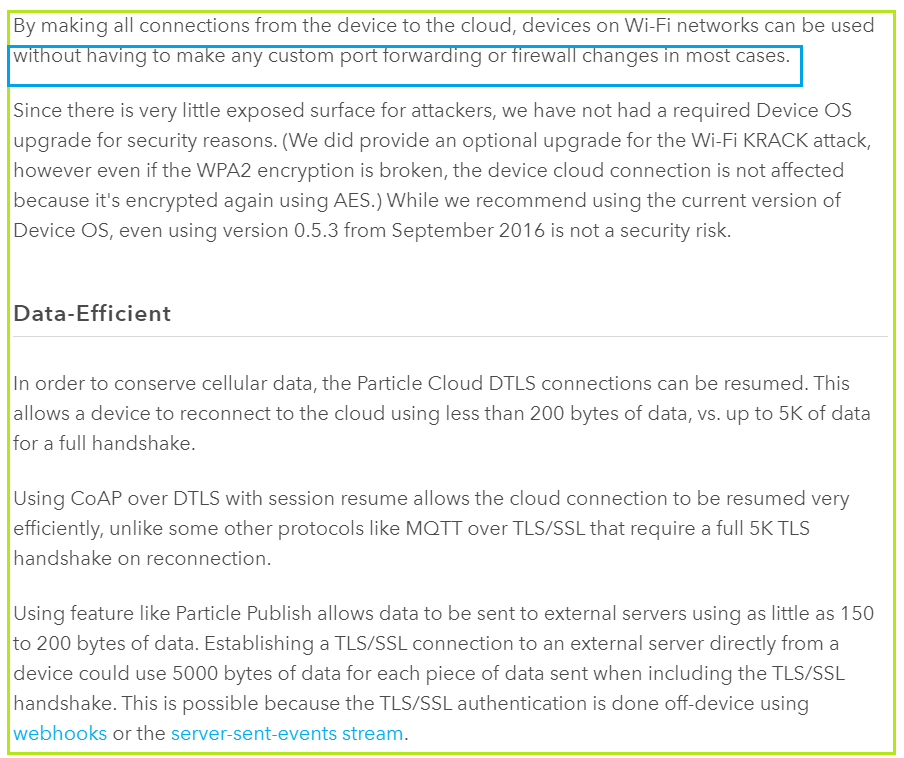


1. Two legged authentication when APP has its own server/database.

App’s server is making API calls to LP server; rather than end – user make the call, which is re-routed through serverless APP.

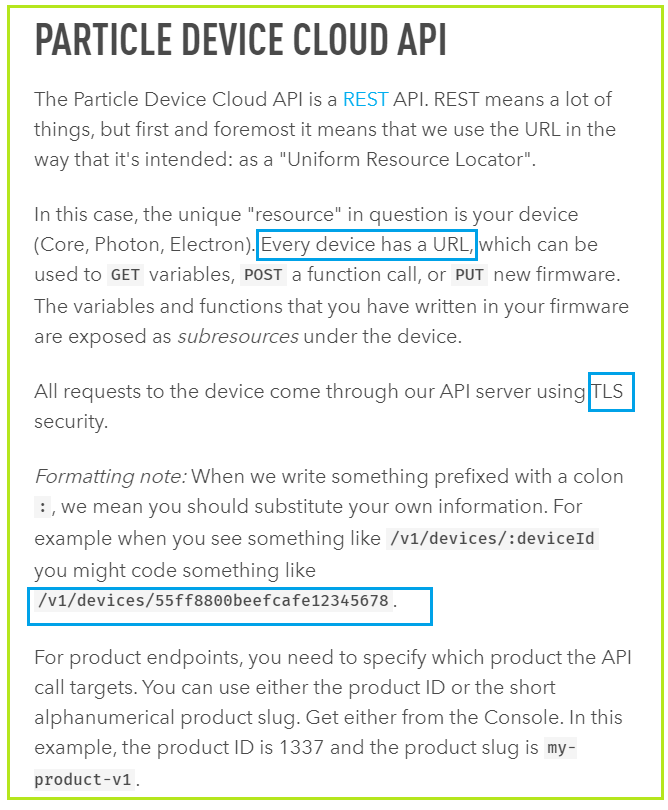
So APP’s server needs client id and credential to authenticate itself, while non-server app really doesn’t need client id and credential.

****

****

**https://www.particle.io/device-cloud/**

1. blallalla

****

1. 7777
2. **Test environment, CI/CD**
3. TBD