Analysis Best Practices

6 Tips for Creating and Deploying Scripts using Analysis using NodeJS

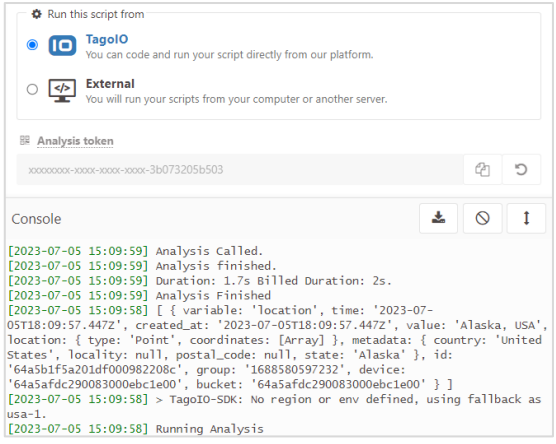
Analysis allows you to manipulate incoming and stored data in realtime. TagoIO currently supports scripts in node.js and python 3 when using our servers (internal mode), but you can use other languages when running it on your own server (external mode). Here are six tips to help you to get the most out of the Analysis tool.

---

1. Use console.log and the Analysis console to output errors

When using Analysis on TagoIO, in internal mode, you can only find errors in your code through the console. When you run scripts on your machine, you may have multiple ways of debugging your code.

That being said, it's very important that you use console.log to output any errors or important information from your Analysis. This will help you track down any error you might run into.



How to:

1. Write in your code:

```
context.log("message here");
```

2. Use the Utils lib from SDK to speed up coding time

The Utils lib from our node.js SDK has different functions to save you time and to help you write quality code.

One of the many basic functions presented in many script examples at TagoIO is the *envToObj* function that transforms environment variables into an easily handled object. You will find many other functions to handle data and get device tokens.

How to:

1. Import the utils lib in your code by:

```
const { Utils } = require("@tago-io/sdk");
```

2. Use the functions

For more information:

https://js.sdk.tago.io/

3. Make use of the Audit log

The Audit log is one of the best developer helper tools that you will find at TagoIO. Through it, you can check if an Analysis was triggered, when, and what event triggered it.

You can also check if you had any problem with the email or sms service.

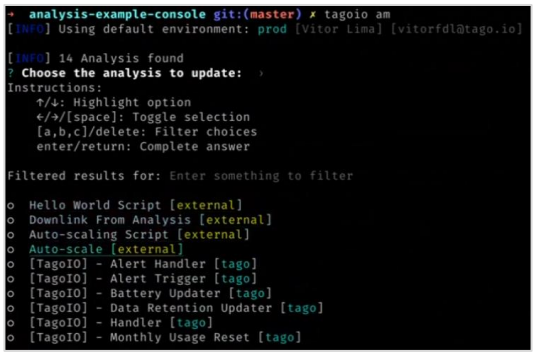
How to:

1. Click on your name at the upper-right corner of the admin.
2. Go to the **Audit Log** option.
3. Select Analysis in **resource type**.
4. Select your analysis in the Analysis dropdown.

4. Make use of the CLI

The TagoIO CLI is very useful for creating and deploying Analysis since it allows users to easily develop analyses in external mode while also encrypting their profile tokens.

After the development process is finished, to deploy their Analysis, all you have to do is run “tagoio deploy” and chose the Analysis you want to deploy. The CLI will automatically build the JavaScript code, send it to TagoIO and enable internal mode.



How to:

1. Open your terminal
2. Install the CLI

    npm i -g @tago-io/cli

3. Install the Analysis-builder

    npm i -g @tago-io/builder

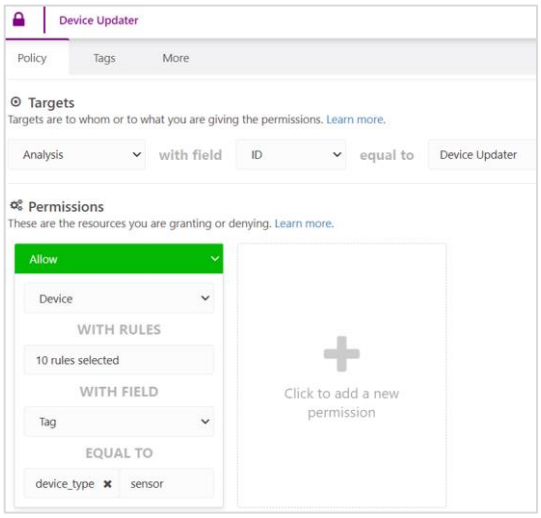
4. Navigate to your project folder
5. Login to the CLI
6. Select the Analyses you want to develop
7. Deploy the finished Analyses to TagoIO

For more information:
https://github.com/tago-io/tagoio-cli#readme

5. Give your Analysis the necessary permissions

In order for your Analyses to work as expected, they need to be granted access to the resources they utilize. This can be done by creating a Policy for your Analysis in the Access Management module.

All you have to do is select your Analysis as the target, select the resources it will use, and specify what actions the Analysis will be able to do with each resource.



Example:

1. Go to **Access Management.**
2. Select the Analysis.
3. Configure the permissions.

6. Make sure you don't receive throughput limit errors

During the execution of your Analysis it's possible to run into a throughput limit error. A throughput limit error can be caused by your Analysis sending too many API requests and overloading it, and this causes the server to end your service.

There are a few ways to avoid this; when sending a larger quantity of requests it's best to use Queues and Pagination. Queues allow you to send batches of requests in a query. Allowing you to send, for example, 5 requests simultaneously without causing a throughput limit.

Example:

```
const device_list = await Resources.devices.listStreaming();
const deleteDevice = async (device) => await
Resources.devices.delete(device.id);
const deleteQueue = queue(deleteDevice, 5);
if (device_list) {
  for await (const devices of device_list) {
    for (const device of devices) {
      void deleteQueue.push(device);
    }
  }
}
if (deleteQueue.started) {
  await deleteQueue.drain();
```

For more information…

Get access to our complete documentation at **https://help.tago.io/portal/**
Join our community and ask questions at **http://community.tago.io/**
Watch our tutorials, webinars, and other videos at **http://tago.io/videos/**
Checkout the Learning Center at **http://tago.io/learning-center/**