Web API Webhooks

Sending central and decentral POST requests

Version 1.1

02.02.2021

Nuki Home Solutions GmbH Münzgrabenstrasse 92/4, 8010 Graz

Introduction

Nuki makes it possible to integrate the Nuki infrastructure to any third party integrator. This empowers any company to implement their business needs in companionship with Nuki devices. To asynchronously inform about any device change in a timely manner Nuki supports webhooks. To create an integration the following prerequisites have to be fulfilled:

- Nuki users have connected their Nuki devices to Nuki servers via a Nuki Bridge
- Nuki users have a Nuki Web account
- The 3rd party integrator has applied for a Nuki Advanced API access and Nuki has activated the Nuki Advanced API

An integrator has to choose between two workflows:

- Central webhooks: Nuki posts webhooks to one distinct URL
- Decentral webhooks: Nuki posts webhooks to several different URLs

There are also two different kind of webhooks regarding the triggering entity:

- Device-triggered webhooks: A webhook created by the Nuki device itself, e.g.: lock action via smartphone app, updated config due to a sync, etc.
- API-triggered webhooks: A webhook as response from an Nuki Advanced API endpoint[1]

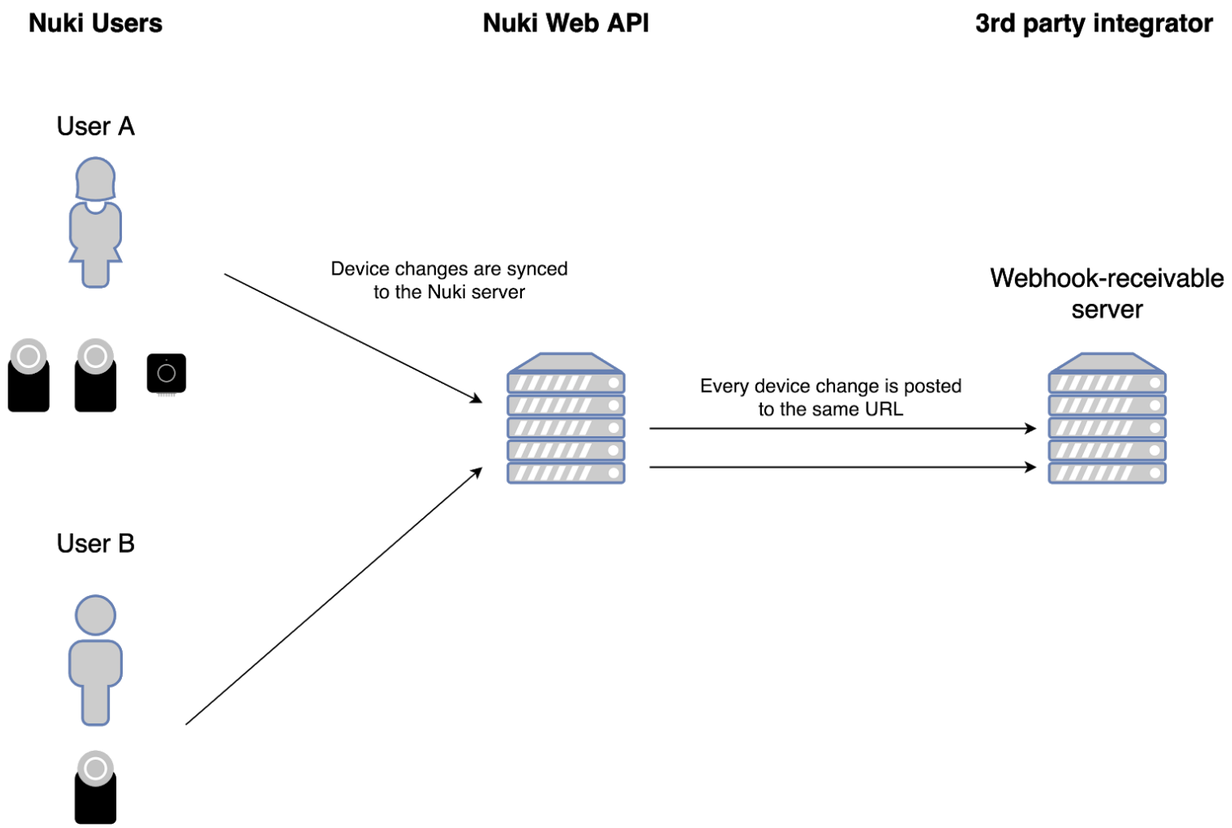
[1] https://api.nuki.io/#!/AdvancedApi

How to integrate

An integrator first has to apply for an Nuki Advanced API integration via https://web.nuki.io/#/pages/web-api. Nuki will audit the appliance and activate the access. This gives the integrator an OAuth "Client ID" and "Client Secret". Secondly a "Redirect Uri" has to be specified. With this three resources ("Client ID", "Client Secret" and "Redirect Uri") an OAuth authentication via Authorization Code Grant[2] is possible. Therefore giving access to the Nuki user's Nuki Web account[3]. To receive webhooks Nuki provides two different implementations based on the needs of an integrator.

Central webhook workflow

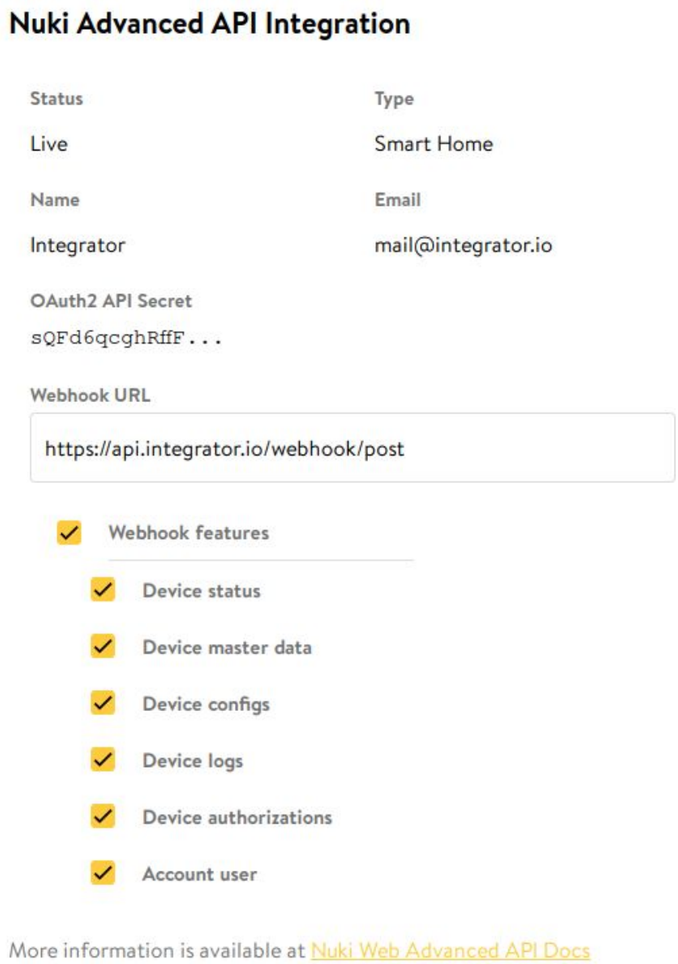
This workflow forwards webhooks to a single URL endpoint. This workflow is best suited for environments with one receiving instance, which handles webhooks for all Nuki users. The next figure explains the basic flow:



[2] https://tools.ietf.org/html/rfc6749#section-4.1
[3] https://developer.nuki.io/page/nuki-web-api-1-3-0/3#heading--code-flow-oauth2

Setup steps for an integrator

1. Configuring the single URL endpoint URL in the "Nuki Advanced Integration" tab at https://web.nuki.io/#/pages/web-api. Checking the device-triggered webhooks to be emitted.



2. Acquiring the Nuki user's agreement to get information about the devices by doing an OAuth code flow[4]. Important is to specify the scope "**webhook.central**", otherwise webhooks are not receivable. Other scopes are optional[5]. After the user's successful authentication a code is returned to the specified request uri. The URL to obtain an OAuth code is constructed as:

```
http://api.nuki.io/oauth/authorize
    ?response_type=code
    &scope=smartlock smartlock.auth account webhook.central
    &client_id=[Client ID from the integration page in Nuki Web]
    &redirect_uri=[Redirect URI from the integration page in Nuki Web]
```

[4] https://tools.ietf.org/html/rfc6749#section-4.1.1
[5] https://developer.nuki.io/page/nuki-web-api-1-3-0/3#heading--scopes

3. To finalize the registration to obtain central webhook an OAuth access token has to be acquired[6]. To obtain an access token a POST request has to be made to:

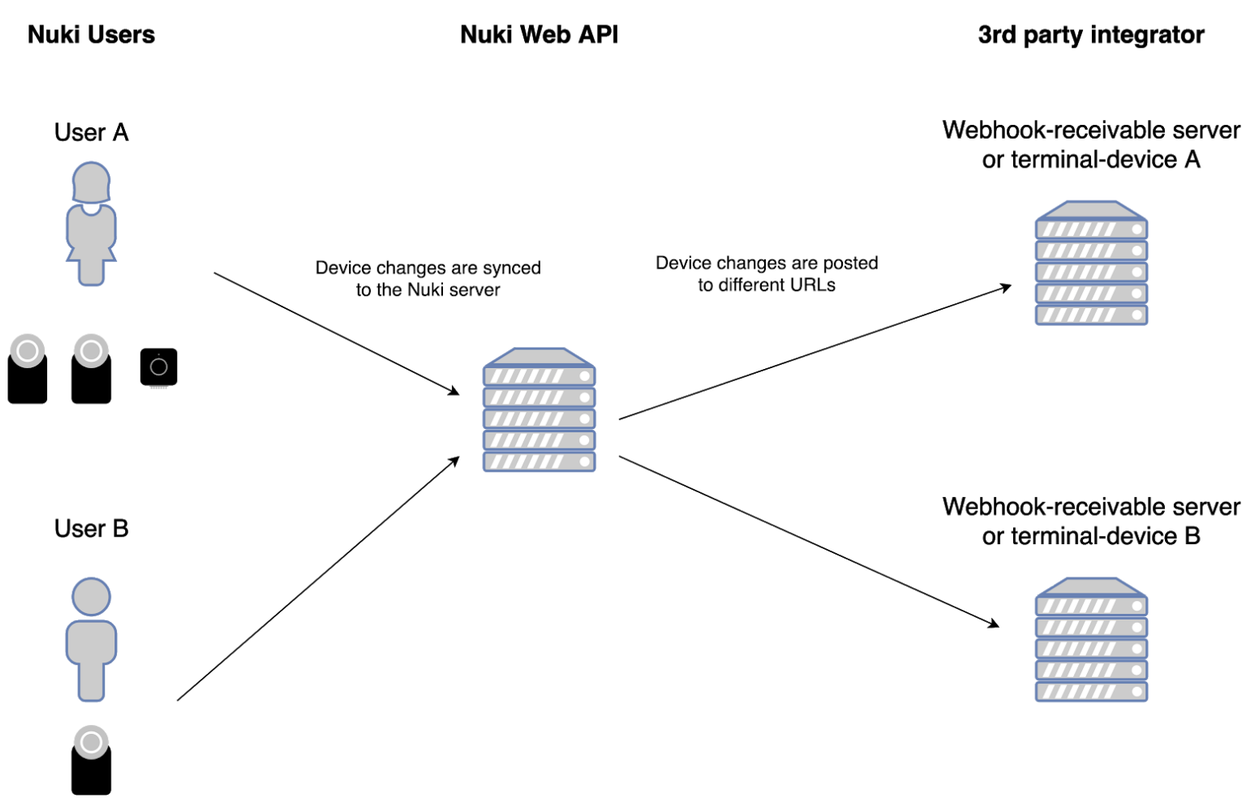
```
POST - http://api.nuki.io/oauth/token?grant_type=authorization_code
    &code=[the code obtained one step ahead]
    &redirect_uri=[Redirect URI from the integration page in Nuki Web]
    &scope=smartlock+smartlock.auth+account+webhook.central
Header:
    • Content-Type: application/x-www-form-urlencoded;charset=UTF-8
    • Authorization: Basic [Base64enc("Client ID":"ClientSecret")]
No Body
```

4. A returned access token ensures a valid central webhook registration. The access token is usable for further actions using the Nuki Web API.

5. Every decentral webhook POST contains a header field "X-Nuki-Signature-SHA256" with the signature value of the signed body's payload. The JSON body is signed with the HMAC SHA256 algorithm based on RFC2104[7], with the secret from the PUT response as the signing key.

Decentral webhook workflow

This workflow is for integrators which want to obtain webhooks on different URLs. This use case is best suited for an integrator which has terminal-devices for each Nuki user separately. The next figure represents the flow graphically:



[6] https://tools.ietf.org/html/rfc6749#section-4.1.3
[7] https://tools.ietf.org/html/rfc2104

Setup steps for an integrator

1. An Advanced API integration is active for the integrator at https://web.nuki.io/#/pages/web-api. The configured webhook url and webhook features are irrelevant, because for decentral webhooks specific configuration steps are needed.
2. Acquiring the Nuki user's agreement to get information about the devices by doing an OAuth code flow[8]. Important is to specify the scope "**webhook.decentral**", otherwise decentral webhooks are not receivable. Other scopes are optional[9]. After the user's successful authentication a code is returned to the specified request uri. The URL to obtain a OAuth code is constructed as:

```
http://api.nuki.io/oauth/authorize
    ?response_type=code
    &scope=smartlock smartlock.auth account webhook.decentral
    &client_id=[Client ID from the integration page in Nuki Web]
    &redirect_uri=[Redirect URI from the integration page in Nuki Web]
```

3. To register a decentral webhook an access token with the scope "webhook.decentral" is required[10]. The access token is usable for further actions using the Nuki Web API. To obtain an access token a POST request has to be made to:

POST - http://api.nuki.io/oauth/token?grant_type=authorization_code
    &code=[*the **code** obtained one step ahead*]
    &redirect_uri=[*Redirect URI from the integration page in Nuki Web*]
    &scope=smartlock+smartlock.auth+account+**webhook.decentral**
Header:
- Content-Type: application/x-www-form-urlencoded;charset=UTF-8
- Authorization: Basic [**Base64enc**("***Client ID***":"***ClientSecret***")]

No Body

4. With the access token it is possible to register a decentral webhook via the Nuki Web API.

PUT https://api.nuki.io/api/decentralWebhook
Header:
- Content-Type: application/json
- Authorization: Bearer [*access token*]

Body:

```
{
   "webhookUrl": "https://decentral123.webhook.at",
   "webhookFeatures": [
      "DEVICE_STATUS", "DEVICE_MASTERDATA", "DEVICE_CONFIG", "DEVICE_LOGS", "DEVICE_AUTHS", "ACCOUNT_USER"
   ]
}
```

The response contains the secret and the identifier of the registered webhook:

```
{
   "id": 123456789,
   "secret": "xxx...",
   "webhookUrl": "https://decentral123.webhook.at",
   "webhookFeatures": [
```

[8] https://tools.ietf.org/html/rfc6749#section-4.1.1
[9] https://developer.nuki.io/page/nuki-web-api-1-3-0/3#heading--scopes
[10] https://tools.ietf.org/html/rfc6749#section-4.1.3

```
      "DEVICE_STATUS", "DEVICE_MASTERDATA", "DEVICE_CONFIG", "DEVICE_LOGS", "DEVICE_AUTHS", "ACCOUNT_USER"
   ]
}
```

5. Every decentral webhook POST contains a header field "X-Nuki-Signature-SHA256" with the signature value of the signed body's payload. The JSON body is signed with the HMAC SHA256 algorithm based on RFC2104[11], with the secret from the PUT response as the signing key.

6. If the PUT request to https://api.nuki.io/api/decentralWebhook is successful decentral webhooks are enabled.

7. To unregister a decentral webhook a DELETE request has to be made to:

DELETE https://api.nuki.io/api/decentralWebhook/{id}

Header:
- Content-Type: application/json
- Authorization: Bearer [*access token*]

8. All registered decentral webhooks can be obtained from:

GET https://api.nuki.io/api/decentralWebhook

Header:
- Content-Type: application/json
- Authorization: Bearer [*access token*]

[11] https://tools.ietf.org/html/rfc2104

How to monitor

Every webhook sent, whether user-triggered or api-triggered, is stored as a log entry. The logs can be accessed via the endpoint https://api.nuki.io/api/key/{apiKeyId}/webhook/logs or can be viewed via Nuki-Web.

Restriction: Only the last 300 messages are stored.

Furthermore we offer an error report. This error report is only sent to the email address of the integrator if the error rate exceeds 5%. A failed attempt is only registered as failed if the HTTP status code of the webhook is not 200 OK, 202 Accepted or 204 No-Content[12].

We may suspend the webhooks service for an URL if the error rate is 100% for an extended period of time.

Description API Endpoint

A webhook is always assigned to exactly one Api Key. To retrieve the logs for a Webhook the following endpoint is provided:

GET https://api.nuki.io/api/key/{apiKeyId}/webhook/logs
```
    ?id=[Optionally filter for older logs]
    &limit=[Amount of logs (default: 50, max: 100)]
```

Header:
- Content-Type: application/json
- Authorization: Bearer [*access token*]

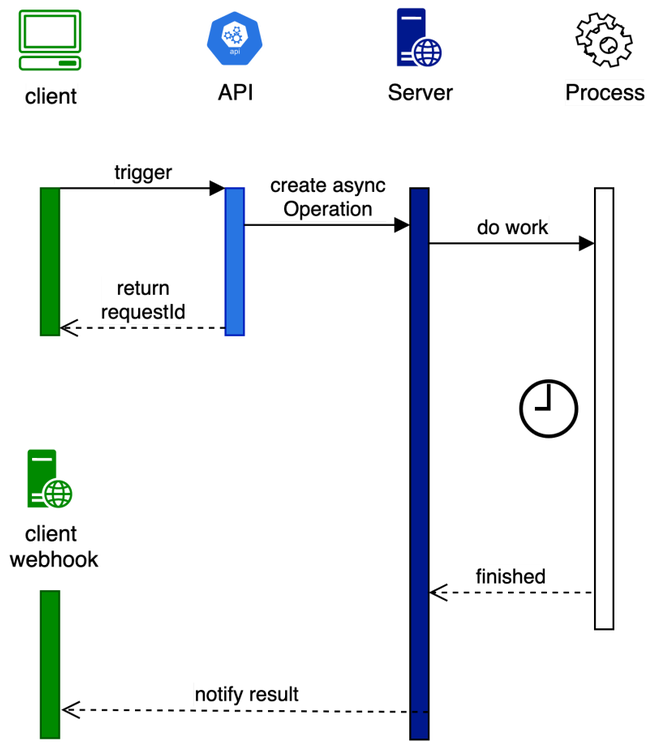
The logs are returned in chronologically descending order. If no parameters are specified, the last max. 50 logs are returned. By specifying the "Id" parameter all logs that were after this specific Id are returned. With the limit parameter you can set the number of logs returned from 1-100 (any other value is ignored and the default value of 50 is assumed). The Id is a hexadecimal representation of an Object ID[13].

[12] https://tools.ietf.org/html/rfc7231#section-6.3
[13] https://docs.mongodb.com/manual/reference/method/ObjectId/

Response Types

The response types are basically divided into 2 types (api- and device-triggered).

Api-Triggered



As shown in the graphic, after calling an Advanced API endpoint, a response is returned synchronously with a request id. At the same time the request is processed asynchronously on the server and returns a result to a user defined webhook after the asynchronous process is finished.

There are two types of result reponses. One is the response for a single device result and the other for multiple devices.

Single Result Response

For *'LockAction'*, *'UnlockAction'* and *'SmartlockAction'*:

WebhookResponse {
  type (WebhookType): type = ['LockAction', 'UnlockAction', 'SmartlockAction'],
  requestId (string): Returned RequestId after Api was triggered,
  smartlockId (long): The Id of the affected device,
  success (boolean): True if the action was performed successfully,
  errorCode (String): [Optional] Indicates the error (hexadecimal representation) in case of failure
}

Smart Lock error codes (in HEX)

Opener error codes (in HEX)

Multiple Result Response

For *'AuthCreation'*:

WebhookResponse {
  inviteCode (string): invite Code
  detail: (DetailItem[])
    smartlockId (long): The Id of the affected device,
    success (boolean): True if the creation was performed successfully,
    errorCode (Integer): [Optional] Indicates the error in case of failure
  ],
  type (WebhookType): type = ['AuthCreation'],
  requestId (string): Responed RequestId after Api was triggereed,
  success (boolean): True if the action was performed successfully,
  errorCode (Integer): [Optional] Indicates the error in case of failure
}

Device-Triggered

The device-triggered Webhooks work like push notifications. These 6 features can be subscribed to

Device status

Is triggered when the status of the device has changed.

StateResponse {
  feature = "DEVICE_STATUS",
  smartlockId (long): The Id of the affected device,
  state (Smartlock.State): Smartlock State Model, Ref: https://api.nuki.io/#!/Smartlock/get,

serverState (int): The server state: 0 .. ok, 1 .. unregistered, 2 .. auth uuid invalid, 3 .. auth invalid, 4 .. offline ,
adminPinState (int): The admin pin state: 0 .. ok, 1 .. missing, 2 .. invalid
}

Smart Lock states

Device master data

Is triggered when the master data of the device has changed or a sync has been triggered.

MasterDataResponse {
feature = "DEVICE_MASTERDATA",
deleted (boolean): flag if the device is deleted or not
smartlockId (long): The Id of the affected device,
accountId (integer): The Id of the affected account,
type (integer): Type of the device
authId (integer): The authorization id ,
name (string): The name of the smartlock ,
favorite (boolean): The favorite flag ,
firmwareVersion (integer, optional, read only): The firmware version ,
hardwareVersion (integer, optional, read only): The hardware version ,
serverState (integer): The server state: 0 .. ok, 1 .. unregistered, 2 .. auth uuid invalid, 3 .. auth invalid, 4 .. offline ,
adminPinState (integer): The admin pin state: 0 .. ok, 1 .. missing, 2 .. invalid ,
creationDate (string, optional): The creation date ,
updateDate (string, optional): The update date
}

Device configs

Is triggered when the config of the device has changed or a sync has been triggered.

ConfigResponse {
feature = "DEVICE_CONFIG",
smartlockId (long): The Id of the affected device,
config (Smartlock.Config): Ref: https://api.nuki.io/#!/Smartlock/get,
advancedConfig (Smartlock.AdvancedConfig): Ref: https://api.nuki.io/#!/Smartlock/get,
openerAdvancedConfig (Smartlock.OpenerAdvancedConfig): Ref: https://api.nuki.io/#!/Smartlock/get,
smartdoorAdvancedConfig (Smartlock.SmartdoorAdvancedConfig): Ref: https://api.nuki.io/#!/Smartlock/get,
}

Device logs

Is triggered when an activity log entry is created.

LogResponse {
feature = "DEVICE_LOGS",
smartlockLog (SmartlockLog): Ref: https://api.nuki.io/#!/SmartlockLog/get_0
}

Lock Actions

trigger (integer): The trigger: 0 .. system, 1 .. manual, 2 .. button, 3 .. automatic, 4 .. web, 5 .. app, 6 .. auto lock, 7 .. accessory, 255 .. keypad
state (integer): The completion state: 0 .. Success, 1 .. Motor blocked, 2 .. Canceled, 3 .. Too recent, 4 .. Busy, 5 .. Low motor voltage, 6 .. Clutch failure, 7 .. Motor power failure, 8 .. Incomplete, 9 .. Rejected, 10 .. Rejected night mode, 254 .. Other error, 255 .. Unknown error

Device authorizations

Is triggered when an authorization is created, modified or deleted.
AuthResponse {
  feature = "DEVICE_AUTHS",
  deleted (boolean): flag if the auth is deleted or not,
  smartlockAuth (SmartlockAuth): Ref: https://api.nuki.io/#!/SmartlockAuth/get_0
}

Account User

Is triggered when an Nuki Web account user is created, modified or deleted.

AccountUserResponse {
  feature = "ACCOUNT_USER",
  deleted (boolean): flag if the account user is deleted or not,
  accountUser (AccountUser): Ref: https://api.nuki.io/#!/AccountUser/get_0
}

Common Use Cases

Activity log notifications

To get notifications via webhook for all new activity log entries the feature "DEVICE_STATUS" has to be set (see Device logs).

The smartlockId references the device for which an activity log entry has been created and the deviceType is added.

deviceTypes:

0 .. Smart Lock
1 .. Box
2 .. Opener
3 .. Smart Door

Lock action entries

See Smart Lock Actions for a list of all possible values for action and Smart Lock states for state and trigger.
The autoUnlock flag indicates if the lockAction was triggered by an Auto Unlock.
name and authId reference the authorization which triggered the lock action.

System entries

System entries in the activity log use the same scheme as actual lock action entries. They have to be identified via the associated “action”.
E.g. a firmware update is identifiable at
action = 243 … firmware update
name = “2.9.0” … updated to firmware version 2.9.0

See the model at https://api.nuki.io/#!/SmartlockLog/get for a list of all possible values for action.

Note: The Activity log notifications depend on the activity log to be activated on the Nuki device. The activity log can be enabled/disabled with the Nuki App or via BLE API which will in both cases lead to an activity log entry, which should be monitored to be aware when webhooks notifications will stop due to missing log entries.
The corresponding actions are
254 .. log enabled
255 .. log disabled

Lock state changes

All lock state changes of a Nuki device can be monitored by setting the feature "DEVICE_STATUS" (see Device status).
Next to the “mode” and “state” values the webhook payload will also contain the last executed lock action and the trigger for it, as well as the current battery state.

See Smart Lock states for a list of all possible values for mode, state, trigger and lastAction.

Note: Theoretically lock state changes can also be monitored via Activity Alog notifications, but those need the activity log to be activated on the Nuki device.

Door state changes

All door state changes of a Nuki device can be monitored by setting the feature "DEVICE_STATUS" (see Device status).

Possible doorState values are:

0 .. unavailable
1 .. deactivated
2 .. door closed
3 .. door opened
4 .. door state unknown
5 .. calibrating

Notes:

- Door states are currently only supported by a Smart Lock 2.0 with activated door sensor.
- Theoretically door state changes can also be monitored via Activity log notifications, but those need the activity log and the door sensor log to be activated on the Nuki device.

Authorization changes

To get notifications via webhooks for all changes on authorizations on a Nuki device, the feature “DEVICE_AUTHS” has to be set (see Device authorizations). This covers all authorizations stored on the Nuki device including Fobs and the Keypad authorization.
The Nuki Bridge and Nuki Web authorizations are theoretically also covered, but deleting or recreating those leads to the Web API stopping to work which also renders the set webhooks useless.

Note: Authorization changes done via Nuki App or the BLE API, as well as authorizations created via direct pairing will only trigger webhooks with the next Nuki Web sync°.

° A Nuki Web sync is done all 24 hours or can be triggered by the Web API via POST /smartlock/{smartlockId}/sync.

Battery warnings

Battery warnings can be retrieved via webhooks by setting the feature "DEVICE_STATUS" (see Device status).

batteryCritical (boolean): True if the battery state of the device is critical

For Smart Locks with current firmware additionally the battery charge state can be retrieved:

batteryCharging (boolean, optional): True if a Nuki battery pack in a Smart Lock is currently charging
batteryCharge (integer, optional): Remaining capacity of a Nuki battery pack in %

Note: Theoretically battery warnings can also be monitored via Activity log notifications, but those need the activity log to be activated on the Nuki device.

Device offline/online

Notifications for devices seen by the Nuki server as offline can be retrieved by setting the feature "DEVICE_STATUS" OR “DEVICE_MASTERDATA” which both contain “serverState” as the value representing the known online/offline state of a device.

Possible serverState values:

0 .. ok
1 .. unregistered
2 .. auth uuid invalid
3 .. auth invalid
4 .. offline

Note: In cases 1-3 the connection of the device to the Nuki Web account needs to be reestablished.

Firmware update

Notifications for changes to the firmware version of a device can be retrieved by setting the feature "DEVICE_MASTERDATA".
The value for firmware Version is provided as an integer which has to be transformed into HEX format to show the current version.

Example:

firmwareVersion = 133135

133135 (DEC) = 2080F (HEX) = FW v. 2.8.15

Admin PIN errors

Changing the Admin PIN on a device also restricts all actions from the Nuki Web (API) for which administration rights are needed. Therefore it is recommended to only change it within the Nuki Web.
Changes done outside of Nuki Web can be retrieved by setting the feature "DEVICE_MASTERDATA". The adminPinState can be used to track if functionality may be missing due to a change here.

Possible values of adminPinState:

0 .. ok
1 .. missing - no Admin PIN set; recommended to add one for security reasons
2 .. invalid - new Admin PIN needs to be set in Nuki Web

Best practices

Test on your devices first

The Nuki Web API can only be tested with real Nuki devices connected to a Nuki Web account.
Follow these steps to develop and test your integration:

- Setup a Nuki device with the Nuki App.
- Connect the device to a Nuki Web account via the Nuki App (a Nuki Web account can be created in the process if not already existing).
- Activate the Web API for the account at https://web.nuki.io/#/pages/web-api.
- Apply for Advanced API access at https://web.nuki.io/#/pages/web-api.
- After we granted you access you can change webhooks settings and URL at https://web.nuki.io/#/pages/web-api / Advanced API integration.
- Create some activity log entries by operating the device.
- Monitor the webhook URL and map payload content according to your needs. Note that changed values are not additionally highlighted in the payload and have to be monitored on your side. More information on values in payloads can be found at our Swagger instance and the Web API documentation.
- Deactivate unnecessary features to reduce the number of webhooks sent to a minimum.

Only select features you need

When testing out the webhook features you will notice that certain settings can create duplicate information. This is done to cover as many use cases as possible while keeping the number of settable options to a minimum. We suggest to start testing out certain use cases, for example the ones described in Common Use Cases.

If you want to explore all possible options you can start with all features activated and clustering information received by the different types, as all webhook payloads contain the feature which triggered them, e.g. feature = "DEVICE_LOGS". From there you can deactivate unneeded features to reduce the number of webhook notifications and duplicates for your integration.

Error handling

If the error rate on a webhook exceeds 5% in the last 24 h, a warning is sent to the email set as contact-point in the application form for the Advanced API. Therefore it is important to set an email which is reachable and will be checked regularly.

In case you change the webhook URL or experience issues on your side, you can also directly monitor sent webhooks via the logs at https://web.nuki.io/#/pages/web-api to be able to quickly track down issues with undelivered webhooks.

Changing the webhook URL can also be done directly via the Web API with POST /api/key/{apiKeyId}/advanced.

See also: How to monitor

Only sync devices if really needed

All state changes and log entries are directly pushed via Bridge to the Nuki server and trigger webhooks immediately.
But for example authorization changes or master data changes done via Nuki App or BLE API will only trigger webhooks with the next Nuki Web sync.
A Nuki Web sync is done all 24 hours or can be triggered by the Web API via POST /smartlock/{smartlockId}/sync, but manual syncing should be avoided as it will drain the devices batteries and can lead to errors when overloading the device or Nuki servers.

When to sync?

- The serverState seems to be not up-to-date.
- An unknown authorization ID in a log entry which needs to be mapped immediately to not break a user interface.

When not to sync?

- In regular intervals “as a general fallback”: Webhooks should be used to avoid polling the Web API and not as an additional feature (to even increase the load on the servers).
- If automated 24-hour synchronization fails regularly (i.e. no "DEVICE_STATUS" webhook for >24 hours.): This is most likely an issue with the Bridge-connection which should be addressed and not covered by more syncing.