REST API overview

SafeDNS Shield filtering configuration is managed through its REST API. The API address is deployment-specific. The default management API port is `8080`, but the host, protocol, and port may be changed during deployment.

Use the following placeholder throughout this guide:

```
<base-url> = http[s]://<shield-management-host>:<management-api-port>
```

For example:

```
http://192.0.2.10:8080
```

Access is normally restricted to source addresses approved during deployment.

Using the API

Authentication

All client-facing API requests require an HTTP bearer token.

```
Authorization: Bearer <token>
Content-Type: application/json
```

Example:

```
curl \
  --header "Authorization: Bearer <token>" \
  --header "Content-Type: application/json" \
  "<base-url>/profiles/"
```

Use the token and management API address assigned to the Shield deployment.

General behavior

- Successfully completed configuration changes take effect immediately. No service restart or separate activation request is required.

- Successful create operations normally return `201 Created`.
- Successful update and retrieval operations normally return `200 OK`.
- Successful deletion and initialization operations return `204 No Content` with no response body.
- Invalid request data may return `422 Unprocessable Entity` with validation details.
- IPv4 addresses are supplied as dotted-decimal strings in request bodies, but some resource paths identify an IPv4 address by its unsigned integer representation.
- The examples use trailing slashes where they appear in the API specification.

Profiles and filtering

A filtering profile combines category rules, application rules, block-page behavior, and several profile-level options.

Profile fields

Field	Type	Description
`id`	integer	Client-supplied profile identifier. Required when creating a profile.
`page_id`	integer or `null`	Identifier of the block page assigned to the profile.
`white_list_only`	boolean	When enabled, only domains on the allowlist and explicitly allowed applications are permitted. Default: `false`.
`empty_dns_answer`	boolean	When enabled, Shield returns an empty DNS answer for a blocked domain instead of directing the client to a block page. Default: `false`.
`appaware_priority_over_bwlists`	boolean	When enabled, application rules take priority over explicit domain allow/deny-list rules. Default: `false`.

A profile request also supports:

Field	Type	Description
`cat_ids`	array of integers	Complete collection of filtering category IDs assigned to the profile.
`apps`	array of objects	Complete collection of application rules assigned to the profile.

Each application rule contains:

```
{
  "app_id": 12,
  "status": "deny"
}
```

Supported statuses are:

- `deny` — blocks domains associated with the application.
- `allow` — explicitly permits resolution of domains associated with the application, including when those domains would otherwise be blocked by the profile's category rules.

Application IDs can be obtained from:

```
GET /app_aware/application/
```

Category IDs are listed in the SafeDNS category reference.

Replacement semantics

When `cat_ids` or `apps` is included in:

```
PATCH /profiles/{profile_id}
```

it replaces the complete existing collection. It is not merged with the current configuration.

- Omitting `cat_ids` leaves the category collection unchanged.
- Sending `"cat_ids": []` removes all category assignments.
- Omitting `apps` leaves the application-rule collection unchanged.
- Sending `"apps": []` removes all application rules.

Initial configuration example

The `/init/` operation creates the initial Shield configuration in one request.

- **Method:** `POST`
- **URL:** `<base-url>/init/`

```
{
  "blockpages": [
    {
      "id": 1,
      "type": 0
    },
    {
```

```
    "id": 2,
    "type": 1
  }
],
"profiles": [
  {
    "profile": {
      "id": 1,
      "page_id": 1,
      "white_list_only": false,
      "empty_dns_answer": false,
      "appaware_priority_over_bwlists": false
    },
    "cat_ids": [3, 4, 12],
    "apps": [
      {
        "app_id": 1,
        "status": "deny"
      },
      {
        "app_id": 12,
        "status": "deny"
      },
      {
        "app_id": 93,
        "status": "deny"
      }
    ]
  }
],
"bw_lists": [
  {
    "profile_id": 1,
    "type": "deny",
    "domains": [
      "example1.com",
      "example2.com",
      "example3.com"
    ]
  }
```

```
  ],
  "napts": [],
  "nets": [
    {
      "ip": "100.100.100.100",
      "profile_id": 1,
      "prefix_len": 32
    },
    {
      "ip": "100.110.110.0",
      "profile_id": 1,
      "prefix_len": 24
    },
    {
      "ip": "100.120.0.0",
      "profile_id": 1,
      "prefix_len": 16
    }
  ],
  "nets6": []
}
```

All six top-level arrays are required. Include an empty array when a resource type is not used:

- `blockpages`
- `profiles`
- `bw_lists`
- `napts`
- `nets`
- `nets6`

The example creates:

1. Two block-page records:
   - Type `0`: default block page.
   - Type `1`: custom block page.
2. Profile `1`, assigned to block page `1`.
3. Three blocked content/security categories.
4. Three denied applications.
5. Three explicitly denied domains.
6. One IPv4 host assignment and two IPv4 subnet assignments.

Custom block-page content is configured outside the Shield REST API. The API only creates and assigns the block-page record.

A successful initialization returns:

```
204 No Content
```

Profile configuration examples

Modifying filtering categories

To change categories assigned to a profile, send the complete desired category collection.

The following request replaces the existing categories of profile `1` with six categories:

- **Method:** `PATCH`
- **URL:** `<base-url>/profiles/1`

```
{
  "cat_ids": [3, 4, 12, 66, 70, 71]
}
```

Because `apps` and `profile` are omitted, only the category collection is changed.

To remove all category assignments:

```
{
  "cat_ids": []
}
```

Replacing application rules for a profile

Application rules can be replaced together through the profile endpoint.

- **Method:** `PATCH`
- **URL:** `<base-url>/profiles/1`

```
{
  "apps": [
    {
      "app_id": 1,
      "status": "deny"
```

```
    },
    {
      "app_id": 12,
      "status": "allow"
    }
  ]
}
```

This request removes any other application rules previously assigned to profile `1`. It denies application `1` and explicitly permits domains associated with application `12`, even if category rules would otherwise block them.

To remove all application rules:

```
{
  "apps": []
}
```

Managing individual application rules

Retrieve the application catalogue

- **Method:** `GET`
- **URL:** `<base-url>/app_aware/application/`

Example response:

```
[
  {
    "id": 1,
    "name": "Application name"
  }
]
```

Retrieve all application rules for a profile

- **Method:** `GET`
- **URL:** `<base-url>/profile/1/app_aware`

Example response:

```
[
  {
```

```
    "profile_id": 1,
    "app_id": 12,
    "status": "allow"
  }
]
```

Add one application rule

- **Method:** `POST`
- **URL:** `<base-url>/profile/1/app_aware`

```
{
  "app_id": 12,
  "status": "allow"
}
```

Add several application rules with the same status

- **Method:** `POST`
- **URL:** `<base-url>/profile/1/app_aware/batch`

```
{
  "app_ids": [1, 12, 93],
  "status": "deny"
}
```

Delete an application rule

The final path value is the application ID. The current OpenAPI parameter name is `app_aware_id`, but it identifies the same value as `app_id`.

- **Method:** `DELETE`
- **URL:** `<base-url>/profile/1/app_aware/12`

No request body is required.

Adding a domain to the allowlist

To exempt a domain from category-based blocking, add it to the allowlist of the relevant profile.

- **Method:** `POST`
- **URL:** `<base-url>/profile/1/bw_list`

```
{
  "type": "allow",
  "domain": "example4.com"
}
```

Adding domains to the denylist

To block several domains directly, use the batch operation.

- **Method:** `POST`
- **URL:** `<base-url>/profile/1/bw_list/batch`

```
{
  "type": "deny",
  "domains": [
    "example5.com",
    "example6.com",
    "example7.com",
    "example8.com",
    "example9.com"
  ]
}
```

Allow/deny-list storage limitation

Shield stores allowlist and denylist domains as hashes. The API therefore cannot retrieve or enumerate the original domain names after they have been submitted.

API clients should retain their own copy of configured allowlists and denylists when later inspection, reconciliation, or synchronization is required.

Update and delete operations still accept the original domain name. Shield calculates its hash internally to locate the stored rule.

Creating a new filtering profile

- **Method:** `POST`
- **URL:** `<base-url>/profiles/`

```
{
  "profile": {
    "id": 2,
    "page_id": 1,
```

```
    "white_list_only": false,
    "empty_dns_answer": false,
    "appaware_priority_over_bwlists": false
  },
  "cat_ids": [3, 4, 12, 13, 66, 70, 71],
  "apps": [
    {
      "app_id": 1,
      "status": "deny"
    },
    {
      "app_id": 12,
      "status": "deny"
    },
    {
      "app_id": 93,
      "status": "deny"
    }
  ]
}
```

This creates profile `2`, assigns block page `1`, enables the listed filtering categories, and adds three denied application rules.

Subscriber assignments

Assigning an IPv4 address or subnet

To assign an IPv4 host to profile `1`, create a `/32` network record.

- **Method:** `POST`
- **URL:** `<base-url>/net/`

```
{
  "ip": "100.100.100.101",
  "profile_id": 1,
  "prefix_len": 32
}
```

To assign a `/24` subnet:

```
{
  "ip": "100.100.101.0",
  "profile_id": 1,
  "prefix_len": 24
}
```

Supported IPv4 prefix lengths are `/10` through `/32`.

Retrieve all IPv4 assignments

- **Method:** `GET`
- **URL:** `<base-url>/net/`

Example response:

```
[
  {
    "ip": "100.100.100.101",
    "profile_id": 1,
    "prefix_len": 32
  },
  {
    "ip": "100.100.101.0",
    "profile_id": 1,
    "prefix_len": 24
  }
]
```

IPv4 integer representation

Individual IPv4 resource paths identify an address by its unsigned 32-bit integer representation.

For example:

```
100.100.100.101 = 1684300901
100.100.101.0   = 1684301056
```

Python conversion example:

```
from ipaddress import IPv4Address


int_ip = int(IPv4Address("100.100.100.101"))
```

```
print(int_ip)  # 1684300901
```

Reassign an IPv4 address or subnet

- **Method:** `PATCH`
- **URL:** `<base-url>/net/1684301056`

```
{
  "ip": "100.100.101.0",
  "profile_id": 2,
  "prefix_len": 24
}
```

Remove an IPv4 address or subnet

- **Method:** `DELETE`
- **URL:** `<base-url>/net/1684301056`

No request body is required.

Assigning an IPv6 address or subnet

- **Method:** `POST`
- **URL:** `<base-url>/net6/`

```
{
  "ip": "2001:db8:100::",
  "profile_id": 1,
  "prefix_len": 64
}
```

Supported IPv6 prefix lengths are `/16` through `/128`.

Retrieve all IPv6 assignments

- **Method:** `GET`
- **URL:** `<base-url>/net6/`

Example response:

```
[
  {
    "ip": "2001:db8:100::",
    "profile_id": 1,
```

```
    "prefix_len": 64
  }
]
```

Identifying subscribers through NAPT port ranges

NAPT records assign different source-port ranges on a shared IPv4 address to different filtering profiles.

Port boundaries are inclusive. A range from `10000` to `19999` includes both port `10000` and port `19999`.

Create a NAPT assignment

- **Method:** `POST`
- **URL:** `<base-url>/napt/`

```
{
  "ip": "203.0.113.10",
  "lower_port_bound": 10000,
  "upper_port_bound": 19999,
  "profile_id": 1
}
```

Port values must be between `0` and `65535`. The complete port range is `0` through `65535`.

Create several NAPT assignments

- **Method:** `POST`
- **URL:** `<base-url>/napt/batch`

```
[
  {
    "ip": "203.0.113.10",
    "lower_port_bound": 10000,
    "upper_port_bound": 19999,
    "profile_id": 1
  },
  {
    "ip": "203.0.113.10",
    "lower_port_bound": 20000,
    "upper_port_bound": 29999,
    "profile_id": 2
```

```
  }
]
```

Retrieve all NAPT assignments for an IPv4 address

The path uses the integer representation of the IPv4 address.

- **Method:** `GET`
- **URL:** `<base-url>/napt/3405803786`

`3405803786` is the integer representation of `203.0.113.10`.

Delete several NAPT assignments

Unlike individual NAPT resource paths, the batch-deletion body uses dotted-decimal IPv4 strings.

- **Method:** `DELETE`
- **URL:** `<base-url>/napt/batch`

```
[
  {
    "ip": "203.0.113.10",
    "lower_port_bound": 10000,
    "upper_port_bound": 19999
  },
  {
    "ip": "203.0.113.10",
    "lower_port_bound": 20000,
    "upper_port_bound": 29999
  }
]
```

`profile_id` is not included in the batch-deletion request.

Profile schedules

A schedule temporarily switches one profile to another during specified periods.

The profile identified in the URL is the source profile. `target_profile_id` identifies the profile used during scheduled periods. A profile cannot target itself.

Supported day formats

Both weekday names and integers are accepted:

Integer	Weekday
`0`	Monday
`1`	Tuesday
`2`	Wednesday
`3`	Thursday
`4`	Friday
`5`	Saturday
`6`	Sunday

Weekday names must be lowercase:

```
monday, tuesday, wednesday, thursday, friday, saturday, sunday
```

Examples in this guide use weekday names for readability.

Times use `HH:MM` format and must fall on 30-minute boundaries, such as `08:00`, `12:30`, or `18:00`. `00:00` represents the start of a day and `24:00` represents the end of a day.

Create a schedule

- **Method:** `POST`
- **URL:** `<base-url>/profile/1/schedule`

```
{
  "target_profile_id": 2,
  "periods": [
    {
      "start": {
        "day": "monday",
        "time": "18:00"
      },
      "end": {
        "day": "tuesday",
        "time": "08:00"
      }
    },
    {
      "start": {
```

```
        "day": 5,
        "time": "00:00"
      },
      "end": {
        "day": 5,
        "time": "24:00"
      }
    }
  ]
}
```

This example switches profile `1` to profile `2` from Monday evening through Tuesday morning and for the whole of Saturday.

Update a schedule

- **Method:** `PATCH`
- **URL:** `<base-url>/profile/1/schedule`

To change the target profile for all existing periods:

```
{
  "target_profile_id": 3
}
```

To replace all existing periods:

```
{
  "periods": [
    {
      "start": {
        "day": "friday",
        "time": "18:00"
      },
      "end": {
        "day": "monday",
        "time": "08:00"
      }
    }
  ]
}
```

When `periods` is supplied, all previously configured periods are deleted and replaced by the submitted collection.

Delete a schedule

- **Method:** `DELETE`
- **URL:** `<base-url>/profile/1/schedule`

No request body is required.

Block pages

Block-page records use two types:

Type	Meaning
`0`	Default block page
`1`	Custom block page

Example creation request:

- **Method:** `POST`
- **URL:** `<base-url>/blockpage/`

```
{
  "id": 2,
  "type": 1
}
```

Custom block-page content is configured outside the Shield REST API.

Revision #21
Created 2026-03-06 08:42:09 UTC by Ryan Lane
Updated 2026-07-22 15:09:15 UTC by Andrew Lem