
deSEC DNS API Documentation

deSEC e.V., Individual Contributors

Nov 23, 2020

1	Quickstart	3
2	Manage Account	5
2.1	Obtain a Captcha	5
2.2	Register Account	5
2.3	Log In	7
2.4	Retrieve Account Information	7
2.5	Password Reset	8
2.6	Password Change	9
2.7	Change Email Address	9
2.8	Delete Account	9
2.9	Log Out	10
2.10	Security Considerations	10
3	Manage Tokens	13
3.1	Token Field Reference	13
3.2	Creating a Token	14
3.3	Modifying a Token	15
3.4	Listing Tokens	15
3.5	Retrieving a Specific Token	16
3.6	Deleting a Token	16
3.7	Security Considerations	16
4	Domain Management	17
4.1	Domain Field Reference	17
4.2	Creating a Domain	19
4.3	Listing Domains	19
4.4	Retrieving a Specific Domain	19
4.5	Deleting a Domain	20
5	Retrieving and Creating DNS Records	21
5.1	RRset Field Reference	21
5.2	Creating an RRset	23
5.3	Retrieving all RRsets in a Zone	24
5.4	Retrieving a Specific RRset	25
5.5	Modifying an RRset	26
5.6	Deleting an RRset	27

5.7	Bulk Operations	27
5.8	Notes	28
6	Configuring your dynDNS Client	31
6.1	Option 1: Use Your Router	31
6.2	Option 2: Use ddclient	32
7	TLS Certificate with Let's Encrypt	35
7.1	certbot with deSEC hook	35
7.2	Other ACME clients	36
8	IP Update API	37
8.1	Update Request	37
8.2	Update Response	38
8.3	Examples	39
9	Endpoint Reference	41
10	Rate Limits	43
11	API Versions and Lifecycle	45
12	Getting Help	47
13	About this document	49

The deSEC DNS API is a REST interface that allows easy management of DNS information. The interface design aims for simplicity so that tasks such as creating domains and manipulating DNS records can be handled with ease and in an intuitive fashion.

Server-side operations, such as creation of domains or DNS records, expect JSON-formatted user input in the body of the POST, PATCH, or PUT request (see below). The request is required to come with a `Content-Type: application/json` header field.

API functionality is demonstrated using the command line tool `curl`. To pretty-print JSON output, process the data through `jq`: `curl ... | jq ..`

Windows users: We are told that the `curl` commands in this documentation sometimes do not work. In this case, try moving the request payload in front of the `curl` call, like this:

```
echo {"email": "mail@example.com", "password": "foobar"} | curl -X POST https://desec.io/api/v1/auth/login/ --header "Content-Type: application/json" --data @-
```


To use our domain management API, you need to register an account with deSEC. Here's a quick intro how to get started:

1. *Obtain a Captcha* and solve it:

```
curl -X POST https://desec.io/api/v1/captcha/
```

Note down the captcha ID from the response body, and figure out the solution from the `challenge` field. It's a base64-encoded PNG image which you can display by directing your browser to the URL `data:image/png;base64,<challenge>`, after replacing `<challenge>` with the value of the challenge response field

2. *Register Account*:

```
curl -X POST https://desec.io/api/v1/auth/ \
  --header "Content-Type: application/json" --data @- <<EOF
{
  "email": "youremailaddress@example.com",
  "password": "yourpassword",
  "captcha": {
    "id": "00010203-0405-0607-0809-0a0b0c0d0e0f",
    "solution": "12H45"
  }
}
EOF
```

Before activating your account, we need to verify your email address. To that end, we will send you an email, containing a validation link of the form `https://desec.io/api/v1/v/activate-account/<code>/`. To confirm your address and activate your account, simply click the link.

3. *Log In*:

```
curl -X POST https://desec.io/api/v1/auth/login/ \
  --header "Content-Type: application/json" --data @- <<< \
  '{"email": "youremailaddress@example.com", "password": "yourpassword"}'
```

The response body will contain an `token` which is used to authenticate requests to the DNS management endpoints as demonstrated in the next step.

4. Create a DNS zone:

```
curl -X POST https://desec.io/api/v1/domains/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"name": "example.com"}'
```

5. Yay! Keep browsing the *Domain Management* section of the docs to see how to continue.

Access to the domain management API is granted to registered and logged in users only. Users can register an account free of charge through the API as described below.

2.1 Obtain a Captcha

Before registering a user account, you need to solve a captcha. You will have to send the captcha ID and solution along with your registration request. To obtain a captcha, issue a POST request as follows:

```
curl -X POST https://desec.io/api/v1/captcha/
```

The response body will be a JSON object with an `id` and a `challenge` field. The value of the `id` field is the one that you need to fill into the corresponding field of the account registration request. The value of the `challenge` field is the base64-encoded PNG representation of the captcha itself. You can display it by directing your browser to the URL `data:image/png;base64,<challenge>`, after replacing `<challenge>` with the value of the `challenge` response field.

Captchas expire after 24 hours. IDs are also invalidated after using them in a registration request. This means that if you send an incorrect solution, you will have to obtain a fresh captcha and try again.

2.2 Register Account

You can register an account by sending a POST request containing your email address, a password, and a captcha ID and solution (see *Obtain a Captcha*), like this:

```
curl -X POST https://desec.io/api/v1/auth/ \
  --header "Content-Type: application/json" --data @- <<EOF
{
  "email": "youremailaddress@example.com",
  "password": "yourpassword",
```

(continues on next page)

(continued from previous page)

```
"captcha": {
  "id": "00010203-0405-0607-0809-0a0b0c0d0e0f",
  "solution": "12H45"
}
}
EOF
```

Please consider the following when registering an account:

- Surrounding whitespace is stripped automatically from passwords.
- We do not enforce restrictions on your password. However, to maintain a high level of security, make sure to choose a strong password. It is best to generate a long random string consisting of at least 16 alphanumeric characters, and use a password manager instead of attempting to remember it.
- If you do not require a password at the moment, you can pass `null` (the JSON value, not the string!). If you create an account this way, it will not be possible to *Log In*. You can set a password later using the *Password Reset* procedure.
- Your email address is required for account recovery in case you forgot your password, for contacting support, etc. We also send out announcements for technical changes occasionally. It is thus deSEC's policy to require users provide a valid email address.

When attempting to register a user account, the server will reply with `202 Accepted`. In case there already is an account for that email address, nothing else will be done. Otherwise, you will receive an email with a verification link of the form `https://desec.io/api/v1/v/activate-account/<code>/`. To activate your account, click on that link (which will direct you to our frontend) or send a `POST` request on the command line. The link expires after 12 hours.

If there is a problem with your email address, your password, or the proposed captcha solution, the server will reply with `400 Bad Request` and give a human-readable error message that may look like:

```
HTTP/1.1 400 Bad Request

{
  "password": [
    "This field may not be blank."
  ]
}
```

2.2.1 Zone Creation during Account Registration

Note: The following functionality is intended for internal deSEC use only. Availability of this functionality may change without notice.

Along with your account creation request, you can provide a domain name as follows:

```
curl -X POST https://desec.io/api/v1/auth/ \
  --header "Content-Type: application/json" --data @- <<EOF
{
  "email": "youremailaddress@example.com",
  "password": "yourpassword",
  "captcha": {
    "id": "00010203-0405-0607-0809-0a0b0c0d0e0f",
    "solution": "12H45"
  },

```

(continues on next page)

(continued from previous page)

```
    "domain": "example.org"
  }
EOF
```

If the `domain` field is present in the request payload, a DNS zone will be created for this domain name once you activate your account using the verification link that we will send to your email address. If the zone cannot be created (for example, because the domain name is unavailable), your account will be deleted, and you can start over with a fresh registration.

2.3 Log In

All interactions with the API that require authentication must be authenticated using a token that identifies the user and authorizes the request. Logging in is the process of obtaining such a token.

In order to log in, you need to confirm your email address first. Afterwards, you can ask the API for a token that can be used to authorize subsequent DNS management requests. To obtain such a token, send a POST request with your email address and password to the `/auth/login/` endpoint:

```
curl -X POST https://desec.io/api/v1/auth/login/ \
  --header "Content-Type: application/json" --data @- <<< \
  '{"email": "youremailaddress@example.com", "password": "yourpassword"}'
```

If email address and password match our records, the server will reply with 200 OK and return the token in the `token` field of the response body:

```
{
  "created": "2018-09-06T09:07:43.762697Z",
  "id": "8f9cbae2-c862-48a4-b3f0-2cb1a80df168",
  "token": "f07Q0TRmEb-CRWPe4h64_iV2jbet",
  "name": "login"
}
```

In case of credential mismatch, the server replies with 401 Unauthorized.

Note: Every time you send a POST request to this endpoint, an additional token will be created. Existing tokens will *remain valid*.

To authorize subsequent requests with the new token, set the HTTP Authorization header to the token value, prefixed with `Token`:

```
curl -X GET https://desec.io/api/v1/ \
  --header "Authorization: Token i-T3b1h_OI-H9ab8tRS98stGtURe"
```

2.4 Retrieve Account Information

To request information about your account, send a GET request to the `/auth/account/` endpoint:

```
curl -X GET https://desec.io/api/v1/auth/account/ \
  --header "Authorization: Token i-T3b1h_OI-H9ab8tRS98stGtURe"
```

A JSON object representing your user account will be returned:

```
{
  "created": "2019-10-16T18:09:17.715702Z",
  "email": "youremailaddress@example.com",
  "id": "9ab16e5c-805d-4ab1-9030-af3f5a541d47",
  "limit_domains": 15
}
```

Field details:

created

Access mode read-only

Registration timestamp.

email

Access mode read-only

Email address associated with the account.

id

Access mode read-only

User ID.

limit_domains

Access mode read-only

Maximum number of DNS zones the user can create.

2.5 Password Reset

In case you forget your password, you can reset it. To do so, send a POST request with your email address and a captcha ID and solution (see *Obtain a Captcha*) to the `/auth/account/reset-password/` endpoint:

```
curl -X POST https://desec.io/api/v1/auth/account/reset-password/ \
  --header "Content-Type: application/json" --data @- <<EOF
  {
    "email": "youremailaddress@example.com",
    "captcha": {
      "id": "00010203-0405-0607-0809-0a0b0c0d0e0f",
      "solution": "12H45"
    }
  }
EOF
```

The server will reply with 202 Accepted. If there is no account associated with this email address, nothing else will be done. Otherwise, you will receive an email with a URL of the form `https://desec.io/api/v1/v/reset-password/<code>/`. To perform the actual password reset, click on that link (which will direct you to our frontend) or send a POST request to this URL, with the new password in the payload:

```
curl -X POST https://desec.io/api/v1/v/reset-password/<code>/ \
  --header "Content-Type: application/json" --data @- <<< \
  '{"new_password": "yournewpassword"}'
```

This URL expires after 12 hours. It is also invalidated by certain other account-related activities, such as changing your email address.

Once the password was reset successfully, we will send you an email informing you of the event.

2.6 Password Change

To change your password, please follow the instructions for *Password Reset*.

2.7 Change Email Address

To change the email address associated with your account, send a POST request with your email address, your password, and your new email address to the `/auth/account/change-email/` endpoint:

```
curl -X POST https://desec.io/api/v1/auth/account/change-email/ \
  --header "Content-Type: application/json" --data @- <<EOF
{
  "email": "youremailaddress@example.com",
  "password": "yourpassword",
  "new_email": "anotheremailaddress@example.net"
}
EOF
```

If the correct password has been provided, the server will reply with `202 Accepted`. In case there already is an account for the email address given in the `new_email` field, nothing else will be done. Otherwise, we will send an email to the new email address for verification purposes. It will contain a link of the form `https://desec.io/api/v1/v/change-email/<code>/`. To perform the actual change, click on that link (which will direct you to our frontend) or send a POST request on the command line.

The link expires after 12 hours. It is also invalidated by certain other account-related activities, such as changing your password.

Once the email address was changed successfully, we will send a message to the old email address for informational purposes.

2.8 Delete Account

Before you can delete your account, it is required to first delete all your domains from deSEC (see *Deleting a Domain*).

To delete your (empty) account, send a POST request with your email address and password to the `/auth/account/delete/` endpoint:

```
curl -X POST https://desec.io/api/v1/auth/account/delete/ \
  --header "Content-Type: application/json" --data @- <<< \
  '{"email": "youremailaddress@example.com", "password": "yourpassword"}'
```

If the correct password has been provided, the server will reply with `202 Accepted` and send you an email with a link of the form `https://desec.io/api/v1/v/delete-account/<code>/`. To finish the deletion, click on that link (which will direct you to our frontend) or send a POST request on the command line.

The link expires after 12 hours. It is also invalidated by certain other account-related activities, such as changing your email address or password.

If your account still contains domains, the server will respond with 409 `Conflict` and not delete your account.

2.9 Log Out

To invalidate an authentication token (log out), send a POST request to the the log out endpoint:

```
curl -X POST https://desec.io/api/v1/auth/logout/ \  
  --header "Authorization: Token i-T3blh_OI-H9ab8tRS98stGtURe"
```

To delete other tokens based on their ID, see *Deleting a Token*.

2.10 Security Considerations

Confirmation Codes Some account-related activities require the user to explicitly reaffirm her intent. For this purpose, we send a link with a confirmation code to the user's email address. Although clients generally should consider these codes opaque, we would like to give some insights into how they work.

The code is a base64-encoded encrypted-then-signed JSON representation of the user's intent. Encryption/decryption and authentication (sign/verify) is handled by [pyca/cryptography's Fernet implementation](#) which is uses AES-CBC and HMAC-SHA256 with specifically derived key material. The HMAC also signs the current time (i.e. when the intent was expressed). During verification, codes are checked for freshness and rejected when older than allowed.

The encoded intent is composed of the user ID and any extra parameters that were submitted along with the intent. An example of such a parameter is the new email address in the context of a *change email address* operation. Parameters that are unknown at code generation time are not included in the code and must be provided via POST request payload when using the code. A typical example of this is the new password in a *password reset* operation, as it is only provided when the code is being used (and not at the time when the code is requested).

In order to prevent race conditions, we augment the code with additional data which we use to invalidate codes when the user state is modified (e.g. by performing another sensitive account operation). This is achieved by including the combined hash of a) the account operation type (e.g. password reset), b) the account's activation status, c) the account's current email address, and d) the user's password hash. When a confirmation code is used, we recompute this hash based on the user's current state, and only perform the requested action if the hash is reproduced identically. If any of these parameters happens to change before a code is applied, the code will be rendered invalid, and the operation will fail. This measure blocks scenarios such as using an old email address change code after a more recent password change. (Note that it is sometimes possible to revert the state so that an old code becomes valid again, such as when you change the email address twice, with the second change undoing the first one. This issue does not occur for password changes; those do permanently invalidate other codes.)

This approach allows us to securely authenticate sensitive user operations without keeping a list of requested operations on the server. This is both an operational and a privacy advantage. For example, if the user expresses her intent to change the account email address, we do not store that new address on the server until the confirmation code is used (from which the new address is then extracted).

Email verification Operations that require verification of a new email address (such as when registering first), the server response does not depend on whether another user is already using that address. This is to prevent clients from telling whether a certain email address is registered with deSEC or not.

Verification emails will only be sent out if the email address is not yet associated with an account. Otherwise, nothing will happen.

Also, accounts are created on the server side when the registration request is received (and kept in inactive state). That is, state exists on the server even before the email address is confirmed. Confirmation merely activates the existing account. The purpose of this is to avoid running the risk of sending out large numbers of emails to the same address when a client decides to send multiple registration requests for the same address. In this case, no emails will be sent after the first one.

Password Security Password information is stored using [Django's default method, PBKDF2](#).

Manage Tokens

To make authentication more flexible, you can create and configure multiple authentication tokens. To that end, we provide a set of token management endpoints that are separate from the login and logout endpoints. The most notable differences are that the login endpoint needs authentication with the user credentials (email address and password) and its purpose is to return a token with a broad range of permissions, whereas the token management endpoints are authenticated using an already issued token, for the purpose of configuring more fine-grained token permissions.

When accessing the token management endpoints using a token without sufficient permission, the server will reply with 403 Forbidden.

3.1 Token Field Reference

A JSON object representing a token has the following structure:

```
{
  "created": "2018-09-06T09:08:43.762697Z",
  "id": "3a6b94b5-d20e-40bd-a7cc-521f5c79fab3",
  "last_used": null,
  "name": "my new token",
  "perm_manage_tokens": false,
  "token": "4pnk7u-NHvrEkFzrhFDRTjGFyX_S"
}
```

Field details:

created

Access mode read-only

Type timestamp

Timestamp of token creation, in ISO 8601 format (e.g. 2018-09-06T09:08:43.762697Z).

id

Access mode read-only

Type UUID

Token ID, used for identification only (e.g. when deleting a token). This is *not* the token value.

last_used

Access mode read-only

Type timestamp (nullable)

Timestamp of when the token was last successfully authenticated, or `null` if the token has never been used.

In most cases, this corresponds to the last time when an API operation was performed using this token. However, if the operation was not executed because it was found that the token did not have sufficient permission, this field will still be updated.

name

Access mode read, write

Type string

Token name. It is meant for user reference only and carries no operational meaning. If omitted, the empty string is assumed.

Certain API operations will automatically populate the `name` field with suitable values such as “login” or “dyn-dns”.

perm_manage_tokens

Access mode read, write

Type boolean

Permission to manage tokens (this one and also all others). A token which does not have this flag set cannot access the `auth/tokens/` endpoints.

token

Access mode read-once

Type string

Token value that is used to authenticate API requests. It is only returned once, upon creation of the token. The value of an existing token cannot be recovered (we store it in irreversibly hashed form). For security details, see *Security Considerations*.

3.2 Creating a Token

To create a new token, issue a POST request to the tokens endpoint:

```
curl -X POST https://desec.io/api/v1/auth/tokens/ \
  --header "Authorization: Token mu4W4MHuSc0HyrGD1h/dnKuZBond" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"name": "my new token"}'
```

Note that the name and other fields are optional. The server will reply with `201 Created` and the created token in the response body:

```
{
  "created": "2018-09-06T09:08:43.762697Z",
  "id": "3a6b94b5-d20e-40bd-a7cc-521f5c79fab3",
  "last_used": null,
  "name": "my new token",
  "perm_manage_tokens": false,
  "token": "4pnk7u-NHvrEkFzrhFDRTjGFyX_S"
}
```

The new token will, by default, possess fewer permissions than a login token. In particular, the `perm_manage_tokens` flag will not be set, so that the new token cannot be used to retrieve, modify, or delete any tokens (including itself).

With the default set of permissions, tokens qualify for carrying out all API operations related to DNS management (i.e. managing both domains and DNS records). Note that it is always possible to use the *Log Out* endpoint to delete a token.

If you require tokens with extra permissions, you can provide the desired configuration during creation:

- `perm_manage_tokens`: If set to `true`, the token can be used to authorize token management operations (as described in this chapter).

If a field is provided but has invalid content, `400 Bad Request` is returned, with error details in the body.

3.3 Modifying a Token

To modify a token, send a `PATCH` or `PUT` request to the `auth/tokens/{id}/` endpoint of the token you would like to modify:

```
curl -X POST https://desec.io/api/v1/auth/tokens/{id}/ \
  --header "Authorization: Token mu4W4MHuSc0HyrGD1h/dnKuZBond" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"name": "my new token"}'
```

The ID given in the URL is the ID of the token that will be modified. Upon success, the server will reply with `200 OK`.

The token given in the `Authorization` header requires the `perm_manage_tokens` permission. If permissions are insufficient, the server will return `403 Forbidden`.

`name` and all other fields are optional. The list of fields that can be given is the same as when *Creating a Token*. If a field is provided but has invalid content, `400 Bad Request` is returned, with error details in the body.

Note: As long as the `perm_manage_tokens` permission is in effect, it is possible for a token to grant and revoke its own permissions. However, if the `perm_manage_tokens` permission is removed, the operation can only be reversed by means of another token that has this permission.

3.4 Listing Tokens

To retrieve a list of currently valid tokens, issue a `GET` request as follows:

```
curl -X GET https://desec.io/api/v1/auth/tokens/ \
  --header "Authorization: Token mu4W4MHuSc0HyrGD1h/dnKuZBond"
```

The server will respond with a list of token objects. Up to 500 items are returned at a time. If you have a larger number of tokens configured, the use of *Pagination* is required.

3.5 Retrieving a Specific Token

To retrieve a list of currently valid tokens, issue a GET request to the token's endpoint:

```
curl -X GET https://desec.io/api/v1/auth/tokens/{id}/ \
  --header "Authorization: Token mu4W4MHuSc0HyrGD1h/dnKuZBond"
```

The response will contain a token object as described under *Token Field Reference*. You can use it to check a token's properties, such as name, timestamps of creation and last use, or permissions.

Note: The response does *not* contain the token value itself!

3.6 Deleting a Token

To delete an existing token by its ID via the token management endpoints, issue a DELETE request on the token's endpoint, replacing `{id}` with the token id value:

```
curl -X DELETE https://desec.io/api/v1/auth/tokens/{id}/ \
  --header "Authorization: Token mu4W4MHuSc0HyrGD1h/dnKuZBond"
```

The server will reply with 204 No Content, even if the token was not found.

If you do not have the token UUID, but you do have the token value itself, you can use the *Log Out* endpoint to delete it.

3.7 Security Considerations

This section is for purely informational. Token length and encoding may change in the future.

Any token is generated from 168 bits of randomness at the server and stored in hashed format (PBKDF2-HMAC-SHA256). Guessing the token correctly or reversing the hash is hence practically impossible.

The token value is represented by 28 characters using a URL-safe variant of base64 encoding. It comprises only the characters A-Z, a-z, 0-9, -, and _ (Base64 padding is not needed as the string length is a multiple of 4.)

Old versions of the API encoded 20-byte tokens in 40 characters with hexadecimal representation. Such tokens are not issued anymore, but remain valid until invalidated by the user.

Domain Management

Domain management is done through the `/api/v1/domains/` endpoint. The following sections describe how to create, list, modify, and delete domains using JSON objects.

All operations are subject to rate limiting. For details, see *Rate Limits*.

4.1 Domain Field Reference

A JSON object representing a domain has the following structure:

```
{
  "created": "2018-09-18T16:36:16.510368Z",
  "keys": [
    {
      "dnskey": "257 3 13 WFR160...",
      "ds": [
        "6006 13 1 8581e9...",
        "6006 13 2 f34b75...",
        "6006 13 3 dfb325...",
        "6006 13 4 2fdcf8..."
      ],
      "flags": 257,
      "keytype": "csk"
    },
    ...
  ],
  "minimum_ttl": 3600,
  "name": "example.com",
  "published": "2018-09-18T17:21:38.348112Z",
  "touched": "2018-09-18T17:21:38.348112Z"
}
```

Field details:

created

Access mode read-only

Timestamp of domain creation, in ISO 8601 format (e.g. 2013-01-29T12:34:56.000000Z).

keys

Access mode read-only

Array with DNSSEC key information. Each entry contains `DNSKEY` and `DS` record contents (the latter being computed from the former), and some extra information. For delegation of DNSSEC-secured domains, the parent domain needs to publish these `DS` records. (This usually involves telling your registrar/registry about those records, and they will publish them for you.)

Notes:

- Keys are returned immediately after domain creation or when retrieving a specific domain. In contrast, when listing all domains, the keys field is omitted for performance reasons.
- The contents of this field are generated from PowerDNS' `cryptokeys` endpoint, see https://doc.powerdns.com/md/httpapi/api_spec/#cryptokeys. We look at each active `cryptokey_resource` (active is true) and then use the `dnskey`, `ds`, `flags`, and `keytype` fields.

minimum_ttl

Access mode read-only

Smallest TTL that can be used in an *RRset*. The value is set automatically by the server.

If you would like to use lower TTL values, you can apply for an exception by contacting support. We reserve the right to reject applications at our discretion.

name

Access mode read, write-once (upon domain creation)

Domain name. Restrictions on what is a valid domain name apply on a per-user basis. In general, a domain name consists of lowercase alphanumeric characters as well as hyphens – and underscores `_` (except at the beginning of the name). The maximum length is 191.

Internationalized domain names (IDN) currently are supported through their Punycode representation only (labels beginning with `xn--`). Converters are available on the net, for example at <https://www.punycoder.com/>.

published

Access mode read-only

Timestamp of when the domain's DNS records have last been published, in ISO 8601 format (e.g. 2013-01-29T12:34:56.000000Z).

As we publish record modifications immediately, this indicates the point in time of the last successful write request to a domain's `rrsets/` endpoint.

touched

Access mode read-only

Timestamp of when the domain's DNS records have last been touched. Equal to the maximum of the domain's `published` field and all *RRset* `touched` values.

This usually is the same as `published`, unless there have been *RRset* write operations that did not trigger publication, such as rewriting an *RRset* with identical values.

4.2 Creating a Domain

To create a new domain, issue a POST request to the `/api/v1/domains/` endpoint, like this:

```
curl -X POST https://desec.io/api/v1/domains/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"name": "example.com"}'
```

Only the name field is mandatory.

Upon success, the response status code will be `201 Created`, with the domain object contained in the response body. If an improper request was sent, `400 Bad Request` is returned. This can happen when the request payload was malformed, or when the requested domain name is unavailable (because it conflicts with another user's zone) or invalid (due to policy, see below).

If you have reached the maximum number of domains for your account, the API responds with `403 Forbidden`.

Restrictions on what is a valid domain name apply. In particular, domains listed on the [Public Suffix List](#) such as `co.uk` cannot be registered. (If you operate a public suffix and would like to host it with deSEC, that's certainly possible; please contact support.) Also, domains ending with `.internal` cannot be registered.

Furthermore, restrictions on a per-user basis may apply. In particular, the number of domains a user can create is limited. If you find yourself affected by this limit although you have a legitimate use case, please contact our support.

4.3 Listing Domains

The `/api/v1/domains/` endpoint responds to GET requests with an array of *domain objects*. For example, you may issue the following command:

```
curl -X GET https://desec.io/api/v1/domains/ \
  --header "Authorization: Token {token}"
```

to retrieve an overview of the domains you own. Domains are returned in reverse chronological order of their creation, and DNSSEC keys are omitted.

The response status code in case of success is `200 OK`. This is true also if you do not own any domains; in this case, the response body will be an empty JSON array.

Up to 500 items are returned at a time. If you have a larger number of domains configured, the use of [Pagination](#) is required.

4.4 Retrieving a Specific Domain

To retrieve a domain with a specific name, issue a GET request with the name appended to the `domains/` endpoint, like this:

```
curl -X GET https://desec.io/api/v1/domains/{name}/ \
  --header "Authorization: Token {token}"
```

This will return only one domain (i.e., the response is not a JSON array).

If you own a domain with that name, the API responds with `200 OK` and returns the domain object in the response body. Otherwise, the return status code is `404 Not Found`.

4.5 Deleting a Domain

To delete a domain, send a `DELETE` request to the endpoint representing the domain. Upon success or if the domain did not exist or was not yours in the first place, the response status code is `204 No Content`.

Retrieving and Creating DNS Records

All DNS information is composed of so-called *Resource Record Sets (RRsets)*. An RRset is the set of all Resource Records of a given record type for a given name. For example, the name `example.com` may have an RRset of type A, denoting the set of IPv4 addresses associated with this name. In the traditional Bind zone file format, the RRset would be written as:

```
<name> IN A 127.0.0.1
<name> IN A 127.0.0.2
...
```

Each of these lines is a Resource Record, and together they form an RRset.

The basic units accessible through the API are RRsets, each represented by a JSON object. The object structure is detailed in the next section.

The relevant endpoints all reside under `/api/v1/domains/{name}/rrsets/`, where `{name}` is the name of a domain you own. When operating on domains that don't exist or you don't own, the API responds with a 404 Not Found status code. For a quick overview of the available endpoints, methods, and operations, see [Endpoint Reference](#).

All operations are subject to rate limiting. For details, see [Rate Limits](#).

5.1 RRset Field Reference

A JSON object representing an RRset has the following structure:

```
{
  "created": "2019-09-18T16:32:16.510368Z",
  "domain": "example.com",
  "subname": "www",
  "name": "www.example.com.",
  "type": "A",
  "records": [
    "127.0.0.1",
    "127.0.0.2"
  ]
}
```

(continues on next page)

```
  ],
  "ttl": 3600,
  "touched": "2020-04-06T09:24:09.987436Z"
}
```

Field details:

created

Access mode read-only

Timestamp of RRset creation, in ISO 8601 format (e.g. 2019-09-18T16:32:16.510368Z).

domain

Access mode read-only

Name of the zone to which the RRset belongs.

Note that the zone name does not follow immediately from the RRset name. For example, the `com` zone contains an RRset of type `NS` for the name `example.com.`, in order to set up the delegation to `example.com`'s DNS operator. The DNS operator's nameserver again has a similar `NS` RRset which, this time however, belongs to the `example.com` zone.

name

Access mode read-only

The full DNS name of the RRset. If `subname` is empty, this is equal to `{name}.`, otherwise it is equal to `{subname}.{name}..`

records

Access mode read, write

Array of record content strings. Please note that when a record value contains a domain name, it is in almost all cases required to add a final dot after the domain name. This applies, for example, to the `CNAME`, `MX`, and `SRV` record types. A typical `MX` value would thus be `10 mx.example.com.` (note the trailing dot).

Please also consider the *[caveat on the priority field](#)*.

The maximum number of array elements is 4091, and the maximum length of the array is 64,000 (after JSON encoding).

Records must be given in presentation format (a.k.a. "BIND" or zone file format). Record values that are not given in canonical form, such as `0:0000::1` will be converted by the API into canonical form, e.g. `::1`. Exact validation and canonicalization depend on the record type.

subname

Access mode read, write-once (upon RRset creation)

Subdomain string which, together with `domain`, defines the RRset name. Typical examples are `www` or `_443._tcp`. In general, a subname consists of lowercase alphanumeric characters as well as hyphens `-`, underscores `_`, and dots `.`. Wildcard name components are denoted by `*`; this is allowed only once at the beginning of the name (see RFC 4592 for details). The maximum length is 178. Further restrictions may apply on a per-user basis.

Note that for subnames to be created, they must be explicitly stated. In particular, the `www` name is not automatically created when assigning an IP address to your domain name (by creating an `A` or `AAAA` record). The same applies for the catch-all mechanism: If you would like a record to apply to all otherwise undefined subdomains, the wildcard subdomain `*` must be explicitly given.

ttl**Access mode** read, write

TTL (time-to-live) value, which dictates for how long resolvers may cache this RRset, measured in seconds. The smallest acceptable value is given by the domain's *minimum TTL* setting. The maximum value is 604800 (one week).

type**Access mode** read, write-once (upon RRset creation)

RRset type (uppercase). We support all [RRset types supported by PowerDNS](#), with the exception of DNSSEC-related types (the backend automatically takes care of setting those records properly). You also cannot access the SOA, see *SOA caveat*.

touched**Access mode** read-only

Timestamp of when the RRset was last touched (same format as `created`). This field reflects the most recent write request to the RRset. It is also updated when the write request does not actually change anything (e.g. overwriting a DNS record with identical values).

5.2 Creating an RRset

To create a new RRset, simply issue a POST request to the `/api/v1/domains/{name}/rrsets/` endpoint, like this:

```
curl -X POST https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"subname": "www", "type": "A", "ttl": 3600, "records": ["127.0.0.1", "127.0.0.2
  ↪"]}'
```

`type`, `records`, and `ttl` are mandatory, whereas the `subname` field is optional.

Upon success, the response status code will be 201 `Created`, with the RRset contained in the response body. If the operation cannot be performed with the given parameters, the API returns 400 `Bad Request`. This can happen, for instance, when there is a conflicting RRset with the same name and type, when not all required fields were provided correctly (such as, when the `type` value was not provided in uppercase), or when the record content is semantically invalid (e.g. when you provide an unknown record type, or an `A` value that is not an IPv4 address).

Note that the values of `type` and `subname` as well as the `records` items are strings, and as such the JSON specification requires them to be enclosed in double quotes (with the quotes being part of the field value); your shell or programming language may require another layer of quotes! By contrast, `ttl` is an integer field, so the JSON value does not contain quotes.

5.2.1 Creating a TLSA RRset

A common use case is the creation of a TLSA RRset which carries information about the TLS certificate used by the server that the domain points to. For example, to create a TLSA RRset for `www.example.com`, you can run:

```
curl -X POST https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<EOF
{
```

(continues on next page)

(continued from previous page)

```
"subname": "_443._tcp.www",
"type": "TLSA",
"ttl": 3600,
"records": ["3 1 1 11501875615d4.....dd122bbf9190"]
}
EOF
```

Note: The subname is prefixed with `_{port}._{transport_protocol}`. For a HTTPS server, this will usually be `_443._tcp` (for an otherwise empty subname), or `_443._tcp.www` for the common `www` domain prefix. For other use cases, the values have to be adapted accordingly (e.g. `_993._tcp` for an IMAPS server).

To generate the TLSA from your certificate, you can use a tool like https://www.huque.com/bin/gen_tlsa. We are planning to provide a tool that is connected directly to our API in the future. For full detail on how TLSA records work, please refer to RFC 6698.

5.2.2 Bulk Creation of RRsets

It is often desirable to create several RRsets at once. This is achieved by sending an array of RRset objects to the `rrsets/` endpoint (instead of just one), like this:

```
curl -X POST https://desec.io/api/v1/domains/{name}/rrsets/ \
--header "Authorization: Token {token}" \
--header "Content-Type: application/json" --data @- <<EOF
[
  {"subname": "www", "type": "A", "ttl": 3600, "records": ["1.2.3.4"]},
  {"subname": "www", "type": "AAAA", "ttl": 3600, "records": ["c0::fefe"]},
  ...
]
EOF
```

This is especially useful for bootstrapping a new domain.

For details about input validation and return status codes, please refer to *Bulk Operations*.

5.3 Retrieving all RRsets in a Zone

The `/api/v1/domains/{name}/rrsets/` endpoint responds to GET requests with an array of *RRset objects*. For example, you may issue the following command:

```
curl -X GET https://desec.io/api/v1/domains/{name}/rrsets/ \
--header "Authorization: Token {token}"
```

to retrieve the contents of a zone that you own. RRsets are returned in reverse chronological order of their creation.

The response status code in case of success is 200 OK. This is true also if there are no RRsets in the zone; in this case, the response body will be an empty JSON array.

5.3.1 Pagination

Up to 500 items are returned at a time. If more than 500 items would match the query, the use of the `cursor` query parameter is required. The first page can be retrieved by sending an empty pagination parameter, `cursor=`.

Once in pagination mode, the URLs to retrieve the next (or previous) page are given in the `Link`: response header. For example:

```
Link: <https://desec.io/api/v1/domains/{domain}/rrsets/?cursor=>; rel="first",
      <https://desec.io/api/v1/domains/{domain}/rrsets/?cursor=:prev_cursor>; rel="prev",
      <https://desec.io/api/v1/domains/{domain}/rrsets/?cursor=:next_cursor>; rel="next"
```

where `:prev_cursor` and `:next_cursor` are page identifiers that are to be treated opaque by clients. On the first/last page, the `Link`: header will not contain a `prev/next` field, respectively.

If no pagination parameter is given although pagination is required, the server will return `400 Bad Request`, along with instructions for pagination.

5.3.2 Filtering by Record Type

To retrieve an array of all RRsets from your zone that have a specific type (e.g. all A records, regardless of subname), augment the previous GET request with a `type` query parameter carrying the desired RRset type like:

```
curl https://desec.io/api/v1/domains/{name}/rrsets/?type={type} \
  --header "Authorization: Token {token}"
```

Query parameters used for filtering are fully compatible with *pagination*.

5.3.3 Filtering by Subname

To filter the RRsets array by subname (e.g. to retrieve all records in the `www` subdomain, regardless of their type), use the `subname` query parameter, like this:

```
curl https://desec.io/api/v1/domains/{name}/rrsets/?subname={subname} \
  --header "Authorization: Token {token}"
```

This approach also allows to retrieve all records associated with the zone apex (i.e. `example.com` where subname is empty), by querying `rrsets/?subname=`.

Query parameters used for filtering are fully compatible with *pagination*.

5.4 Retrieving a Specific RRset

To retrieve an RRset with a specific name and type from your zone (e.g. the A record for the `www` subdomain), issue a GET request with the `subname` information and the type appended to the `rrsets/` endpoint, like this:

```
curl https://desec.io/api/v1/domains/{name}/rrsets/{subname}/{type}/ \
  --header "Authorization: Token {token}"
```

This will return only one RRset (i.e., the response is not a JSON array). The response status code is `200 OK` if the requested RRset exists, and `404 Not Found` otherwise.

5.4.1 Accessing the Zone Apex

Note: The RRset at the zone apex (the domain root with an empty subname) *cannot* be queried via `/api/v1/domains/{name}/rrsets//{type}/`. This is due to normalization rules of the HTTP specification which cause the double-slash `//` to be replaced with a single slash `/`, breaking the URL structure.

To access an RRset at the root of your domain, we reserved the special subname value @. This is a common placeholder for this use case (see RFC 1035). As an example, you can retrieve the IPv4 address(es) of your domain root by running:

```
curl https://desec.io/api/v1/domains/{name}/rrsets/@/A/ \
  --header "Authorization: Token {token}"
```

Pro tip: If you like to have the convenience of simple string expansion in the URL, you can add three dots after {subname}, like so:

```
curl https://desec.io/api/v1/domains/{name}/rrsets/{subname}.../{type}/ \
  --header "Authorization: Token {token}"
```

With this syntax, the above-mentioned normalization problem does not occur, and no special treatment is needed for accessing the zone apex. You can think of the three dots as abbreviating the rest of the DNS name.

5.5 Modifying an RRset

To modify an RRset, use the endpoint that you would also use to retrieve that specific RRset. The API allows changing the values of `records` and `tTL`. When using the `PATCH` method, only fields you would like to modify need to be provided. In contrast, if you use `PUT`, the full resource must be specified (that is, all fields, including `subname` and `type`). Examples:

```
curl -X PUT https://desec.io/api/v1/domains/{name}/rrsets/{subname}/{type}/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<EOF
{
  "subname": "{subname}",
  "type": "{type}",
  "ttl": 3600,
  "records": ["..."]
}
EOF

curl -X PATCH https://desec.io/api/v1/domains/{name}/rrsets/{subname}/{type}/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"ttl": 86400}'
```

If the RRset was updated successfully, the API returns `200 OK` with the updated RRset in the response body. If the operation cannot be performed with the given parameters, the API returns `400 Bad Request`. This can happen, for instance, when there is a conflicting RRset with the same name and type, when not all required fields were provided correctly (such as, when the `type` value was not provided in uppercase), or when the record content is semantically invalid (e.g. when you provide an unknown record type, or an `A` value that is not an IPv4 address).

To modify an RRset at the zone apex (empty subname), use the special subname value @ (read more about [Accessing the Zone Apex](#)).

5.5.1 Bulk Modification of RRsets

It is sometimes desirable to modify several RRsets at once. This is achieved by sending an array of RRset objects to the `rrsets/` endpoint (instead of just one), like this:

```
curl -X PUT https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<EOF
  [
    {"subname": "www", "type": "A", "ttl": 3600, "records": ["1.2.3.4"]},
    {"subname": "www", "type": "AAAA", "ttl": 3600, "records": ["c0::fefe"]},
    ...
  ]
EOF
```

Each given RRset is uniquely identified by its subname and type (with subname defaulting to the empty string if omitted). For ttl and records, the usual validation rules apply.

For details about input validation and return status codes, please refer to *Bulk Operations*.

5.6 Deleting an RRset

To delete an RRset, you can send a DELETE request to the endpoint representing the RRset. Alternatively, you can modify it and provide an empty array for the records field ([]).

Upon success or if the RRset did not exist in the first place, the response status code is 204 No Content.

5.6.1 Bulk Deletion of RRsets

It is sometimes desirable to delete an RRset while creating or modifying another one. This is achieved by sending a bulk request with an RRset that has an empty records list [], using the PATCH or PUT method:

```
curl -X PATCH https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<EOF
  [
    {"subname": "www", "type": "A", "ttl": 3600, "records": ["1.2.3.4"]},
    {"subname": "www", "type": "AAAA", "records": []}
  ]
EOF
```

For details about input validation and return status codes, please refer to *Bulk Operations*.

5.7 Bulk Operations

The rrsets/ endpoint supports bulk operations via the POST, PATCH, and PUT request methods. You can simply send an array of RRset objects (instead of just one), like this:

```
curl -X PATCH https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<EOF
  [
    {"subname": "www", "type": "A", "ttl": 3600, "records": ["1.2.3.4"]},
    {"subname": "www", "type": "AAAA", "ttl": 3600, "records": ["c0::fefe"]},
    {"subname": "backup", "type": "MX", "records": []},
    ...
  ]
EOF
```

(continues on next page)

```
    ]
EOF
```

Note that the zone apex is referred to by an empty subname string, "subname": "". (The special character @ is not accepted as an alias.) For context, see [Accessing the Zone Apex](#).

5.7.1 Atomicity

Bulk operations are performed atomically, i.e. either all given RRsets are accepted and published in (or deleted from) the DNS, or none of them are.

This allows you to smoothly apply large DNS changes to your domain *without* running into the undesirable situation of an error showing up half-way through the process when some changes already have been applied.

5.7.2 Field requirements

In all cases, the `subname` field is optional. If missing, the empty subname is assumed.

For the `POST` and `PUT` methods, all other fields are required for each given RRset. With `POST`, only new RRsets are acceptable (i.e. the domain must not yet have an RRset with the same subname and type), while `PUT` allows both creating new RRsets and modifying existing ones.

For the `PATCH` method, only `type` is required; if you want to modify only `tTL` or `records`, you can skip the other field. To create a new RRset using `PATCH`, all fields but `subname` must be specified.

To delete an RRset during a bulk operation, use `PATCH` or `PUT` and set `records` to `[]`.

5.7.3 Input validation

There are two stages of input validation:

1. Sanity checks, such as syntax, basic semantics (e.g. negative TTL), and uniqueness checks. (We both check for uniqueness with respect to pre-existing RRsets as well as with respect to other RRsets sent in the same bulk request.)
2. DNS conformity checks, such as whether the given type is a supported record type, and whether the given record contents are consistent with the type.

If an error occurs in the first validation stage, the request is aborted, and the error(s) are returned. Only if no error occurred, will the request be allowed to proceed to the second stage.

In the first stage, errors are presented as a list of errors, with each list item referring to one part of the bulk request, in the same order. Parts that did not cause errors have an empty error object `{}`, and parts with errors contain more details describing the error. Unfortunately, in step 2, the API currently does not associate the error message with the RRset that caused it.

The successive treatment of stages means that one bulk part with a stage-2 error may appear valid (`{}`) as long as another RRset has a stage-1 error. Only after the stage-1 error is resolved, the request will reach stage 2, at which point an error may appear due to a bulk part that previously seemed valid.

5.8 Notes

Consider the following general remarks that apply to our API as a whole:

- All operations are performed on RRsets, not on the individual Resource Records.
- The TTL (time-to-live: time for which resolvers may cache DNS information) is a property of an RRset (and not of a record). Thus, all records in an RRset share the record type and also the TTL. (This is actually a requirement of the DNS specification and not an API design choice.)
- We have not done extensive testing for reverse DNS, but things should work in principle. If you encounter any problems, please let us know.

Generally, the API supports all [RRset types supported by PowerDNS](#), with a few exceptions for such record types that the backend manages automatically. Thus, these restrictions are not limitations from a practical point of view. Furthermore, special care needs to be taken with some types of records, as explained below.

5.8.1 Restricted Types

ALIAS, DNAME These record types are used very rarely in the wild. Due to conflicts with the security guarantees we would like to give, these record types are disabled in our API. If you attempt to create such RRsets, you will receive a `400 Bad Request` response. In case you have a good reason for using these record types, shoot us an email and we can discuss your case.

DNSKEY, NSEC3PARAM, RRSIG These record types are meant to provide DNSSEC-related information in order to secure the data stored in your zones. RRsets of this type are generated and served automatically by our nameservers. However, you can neither read nor manipulate these RRsets through the API. When attempting such operations, `403 Forbidden` or `400 Bad Request` is returned, respectively.

SOA record The SOA record cannot be read or written through this interface. When attempting to create, modify or otherwise access an SOA record, `400 Bad Request` or `403 Forbidden` is returned, respectively.

The rationale behind this is that the content of the SOA record is entirely determined by the DNS operator, and users should not have to bother with this kind of metadata. Upon zone changes, the backend automatically takes care of updating the SOA record accordingly.

If you are interested in the value of the SOA record, you can retrieve it using a standard DNS query.

5.8.2 Caveats

Record types with priority field The deSEC DNS API does not explicitly support structured records fields (such as the priority field used for MX, SRV and the like).

Instead, those fields are expected to be concatenated in the conventional order used for zone files, with spaces in between them. For MX RRsets, that means that the priority is located at the beginning of the record content, separated from the rest of it by a space (e.g. `10 mx.example.com.`).

CNAME record

- The record value must be terminated by a dot `.` (as in `example.com.`).
- If you create a CNAME record, its presence will cause other RRsets of the same name to be hidden (“occluded”) from the public (i.e. in responses to DNS queries). This is per RFC 1912.

However, as far as the API is concerned, you can still retrieve and manipulate those additional RRsets. In other words, CNAME-induced hiding of additional RRsets does not apply when looking at the zone through the API.

- It is currently possible to create a CNAME RRset with several records. However, this is not legal, and the response to queries for such RRsets is undefined. In short, don’t do it.
- Similarly, you are discouraged from creating a CNAME RRset for the zone apex (main domain name, empty subname). Doing so will most likely break your domain (for example, any NS records that are present

will disappear from DNS responses), and other undefined behavior may occur. In short, don't do it. If you are interested in aliasing the zone apex, consider using an `ALIAS` RRset.

MX record The MX record value consists of the priority value and a mail server name, which must be terminated by a dot `.`. Example: `10 mail.a4a.de.`

NS record

- The record value must be terminated by a dot `.` (as in `ns1.desec.io.`).
- The use of wildcard RRsets (with one component of `subname` being equal to `*`) of type NS is **discouraged**. This is because the behavior of wildcard NS records in conjunction with DNSSEC is undefined, per RFC 4592, Sec. 4.2.

TXT record The contents of the TXT record must be enclosed in double quotes. Thus, when POSTing to the API, make sure to do proper escaping etc. as required by the client you are using. Here's an example of how to create a TXT RRset:

```
curl -X POST https://desec.io/api/v1/domains/{name}/rrsets/ \
  --header "Authorization: Token {token}" \
  --header "Content-Type: application/json" --data @- <<< \
  '{"type": "TXT", "records": ["\"test value1\"", "\"value2\""], "ttl": 3600}'
```

Configuring your dynDNS Client

Here's how to configure your client to send your IP address to our servers so that we can publish it in the DNS. Depending on your use case, one of the following options might be easier than the others.

To update your dynDNS IP address, there are several options:

6.1 Option 1: Use Your Router

For most folks, using the integrated dynDNS client of their router will be easiest. Here are two ways how to configure it.

6.1.1 Use your router's deSEC provider

Some routers have support for deSEC out of the box, and you just need to select the right option (“deSEC”, “desec.io”, “dedyn”, or similar). For example, if you run a router with the OpenWRT operation system, watch out for the “desec.io” provider.

6.1.2 Custom Configuration

If your router does not have deSEC preconfigured, the configuration procedure will depend on the specific type of router which is why we can't provide a tutorial for all of them. However, most of the time it boils down to enter the following details in your router configuration:

- Update Server `update.dedyn.io`, or Update URL `https://update.dedyn.io/`
- Username (the full name of the domain you want to update, e.g. `yourname.dedyn.io`)
- Hostname (same as your username)
- Token (long random string for authorization)

Advanced API users: The dynDNS token technically is a regular API token. You can also use the token to make requests to our REST API. (Currently, all tokens are equally powerful, i.e. a token used for dynDNS updates can also be used to perform other kinds of API operations. Token scoping is on our roadmap.)

IPv6 Support

There is a chance that your router already properly supports pushing its IPv6 address to us. If it does not, you can try to let our servers determine your IPv6 address by using IPv6 to connect. To see if this method works for you, modify the “Update Server” or “Update URL” setting in your router’s configuration to `update6.dedyn.io` and `https://update6.dedyn.io/`, respectively.

Note that when using this update server, your IPv4 address will be deleted from the DNS, and your domain will operate in IPv6-only mode. (For an explanation why that is the case, see [Determine IP addresses](#).) It is **not** possible to set up IPv4 and IPv6 by using both update servers in an alternating fashion.

To update both your IPv4 and IPv6 address at the same time, most routers need to be configured with an update URL that provides both IP addresses via query string parameters, e.g. `https://update.dedyn.io/?myipv4=1.2.3.4&myipv6=fd08::1234`, and provide placeholders for the respective addresses. To find out the placeholder names for your router, please refer to the manual of your device.

Example: Fritz!Box Devices

For Fritz!Box devices, for example, the respective URL reads: `https://update.dedyn.io/?myipv4=<ipaddr>&myipv6=<ip6addr>`.

Field	Entry
DynDNS Provider	User-defined
Update URL ¹	<code>https://update.dedyn.io/?myipv4=<ipaddr>&myipv6=<ip6addr></code>
Domain Name ²	<code><your domain>.dedyn.io</code>
Username ³	<code><your domain>.dedyn.io</code>
Password ⁴	<code><your authorization token></code>

Note 1 Note that the placeholders `<ipaddr>` and `<ip6addr>` in the update URL must remain unchanged; your router will substitute them automatically. Furthermore, it is neither necessary nor recommended to use the placeholders `<username>` and `<passwd>` as the Fritz!Box makes use of HTTP basic authentication, see [IP Update Authentication](#).

Note 2 This entry is not used by deSEC - you only need to enter it as Fritz!Box mandates it

Note 3 **Not** your deSEC username! Instead, use the domain you want to update, see [IP Update Authentication](#) for details.

Note 4 A valid access token for the domain. **Not** you deSEC account password!

6.2 Option 2: Use ddclient

6.2.1 Automatic configuration (Debian-/Ubuntu-based systems)

If you’re on Debian, Ubuntu or any other Linux distribution that provides you with the `ddclient` package, you can use it to update your IP address with our servers. Note that depending on the `ddclient` version you are using, IPv6 support may be limited.

To install `ddclient`, run `sudo apt-get install ddclient`. If a configuration dialog does not appear automatically, use `sudo dpkg-reconfigure ddclient` to start the configuration process.

In the configuration process, select “other” dynamic DNS service provider, and enter `update.dedyn.io` as the dynamic DNS server. Next, tell `ddclient` to use the “dyndns2” protocol to perform updates. Afterwards, enter the username and the token that you received during registration. Last, tell `ddclient` how to detect your IP address, your domain name and the update interval.

Note: As of the time of this writing, `ddclient` does not use an encrypted HTTPS connection by default. To enable it, open `/etc/ddclient.conf` and add `ssl=yes` above the `server=` statement. We **strongly recommend** doing so; otherwise, your credentials will be exposed during transmission.

6.2.2 Manual configuration (other systems)

After installing `ddclient`, you can start with a `ddclient.conf` configuration file similar to this one, with the three placeholders replaced by your domain name and your token:

```
protocol=dyndns2
# "use=cmd" and the curl command is one way of doing this; other ways exist
use=cmd, cmd='curl https://checkip4.dedyn.io/'
ssl=yes
server=update.dedyn.io
login=[domain]
password='[token] '
[domain]
```

For more information, check out [these two sections](#) of the `ddclient` documentation.

Hint: We have been told that in newer versions of `ddclient`, IPv6 can be enabled by replacing `use` with `usev6`, `checkipv4.dedyn.io` with `checkipv6.dedyn.io`, and `update.dedyn.io` with `update6.dedyn.io`. Unfortunately, there seems to be no documentation of the `usev6` setting, so we don’t know if it is reliable. If you know more about this, please open an issue or pull request at <https://github.com/desec-io/desec-stack/>.

To test your setup, run `sudo ddclient -force` and see if everything works as expected.

TLS Certificate with Let's Encrypt

7.1 certbot with deSEC hook

dynDNS by deSEC supports the DNS challenge protocol to make it easy for you to obtain certificates for your domain name easily from anywhere. All you need is `certbot`, your credentials and our certbot hook script. As always, we appreciate your feedback. Shoot us an email!

To obtain a Let's Encrypt Certificate for your dedyn.io domain, follow these steps.

1. **Install Certbot.** There are many ways to install certbot, depending on your distribution and preference. Please follow the official instructions at <https://certbot.eff.org/>.
2. **Install hook script.** To authenticate your dedyn.io domain against Let's Encrypt using the DNS challenge mechanism, you will need to update your domain according to instructions provided by Let's Encrypt. Our hook script automatizes this process for you. To use it, download the following two files and place them into a directory of your choice. Make sure to change the owner/permissions of the file (`chown/chmod`), so that it is only readable by your certbot user (usually `root`).

```
wget https://raw.githubusercontent.com/desec-io/certbot-hook/master/hook.sh
wget https://raw.githubusercontent.com/desec-io/certbot-hook/master/.dedynauth
```

3. **Get a token.** you need to obtain a token for using the API. To get the token use the email address you used when setting up your desec.io account. To get a token run the following curl command:

```
curl -X POST https://desec.io/api/v1/auth/login/ -header "Content-Type: application/json"
  -data @- <<< '{"email": "your-email-address@example.com", "password": "your-desec.io-
account-password-here"}'
```

4. **Configuration.** You need to provide your dedyn.io credentials to the hook script, so that it can write the Let's Encrypt challenge to the DNS on your behalf. To do so, edit the `.dedynauth` file to look something like:

```
DEDYN_TOKEN=[your token] # remove brackets, token from above step
DEDYN_NAME=[yourdomain.example.com] # remove brackets, add your domain to your_
↪desec.io account first
```

5. **Run certbot.** To obtain your certificate, run certbot in manual mode as follows. (For a detailed explanation, please refer to the certbot manual.) Please notice that you need to insert your domain name one more time. (Also, for users not familiar with shell commands, please note that you need to remove the \ if you reformat the command to fit on one line.)

```
certbot --manual --manual-auth-hook ./hook.sh --manual-cleanup-hook ./hook.sh \  
--preferred-challenges dns -d "YOURDOMAINNAME.dedyn.io" certonly
```

You can also use certbot to get wildcard certificates like so:

```
certbot --manual --manual-auth-hook ./hook.sh --manual-cleanup-hook ./hook.sh --preferred-  
challenges dns -d "example.com" -d "*.*example.com" certonly
```

to make the process headless you can add `--manual-public-ip-logging-ok -n`.

Depending on how you installed certbot, you may need to replace `certbot` with `./certbot-auto` (assuming that the `certbot-auto` executable is located in the current directory). Please also note that the hook script may wait up to two minutes to be sure that the challenge was correctly published.

Note: To include subdomains in your certificate, you can specify the `-d` argument several times, e.g. `-d "YOURDOMAINNAME.dedyn.io" -d "www.YOURDOMAINNAME.dedyn.io"`.

If you would like to help improve this hook script, please check out our open issues at <https://github.com/desec-utils/certbot-hook/issues>. We'd highly appreciate your help!

7.2 Other ACME clients

There are other ACME clients that support deSEC out of the box. We currently know of the following:

- [acme.sh](#)

In case you want to dig deeper, here are the details on how our IP update API works. We provide this API to be compatible with most dynDNS clients. However, we also provide a RESTful API that is more powerful and always preferred over the legacy interface described here.

8.1 Update Request

An IP updates is performed by sending a GET request to `update.dedyn.io` via HTTP or HTTPS. The path component can be chosen freely as long as it does not end in `.ico` or `.png`.

You can connect via IPv4 or IPv6. To enforce IPv6, use `update6.dedyn.io`.

Please be aware that while we still accept unencrypted requests, we **urge** you to use HTTPS. For that reason, we also send an HSTS header on HTTPS connections.

8.1.1 IP Update Authentication

You can authenticate your client in several ways. If authentication fails, the API will return a 401 Unauthorized status code.

Preferred method: HTTP Basic Authentication (with token)

Encode your username and token (provided during registration) in the `Authorization: Basic ...` header. This is the method virtually all dynDNS clients use out of the box.

Important: If you dynDNS client asks for a *password*, do not enter your account password (if you have one). Instead, enter your token!

HTTP Token Authentication

Send an `Authorization: Token ...` header along with your request, where `...` is the token issued at registration (or manually created later).

Query string method (discouraged)

Set the `username` and `password` query string parameters (`GET ?username=...&password=...`).

Important: We **strongly discourage** using this method as it comes with a subtle disadvantage: We log all HTTP request URLs for a few days to facilitate debugging. As a consequence, this method will cause your secret token to end up in our log files in clear text. The method is provided as an emergency solution where folks need to deal with old and/or crappy clients. If this is the case, we suggest looking for another client.

8.1.2 Determine Hostname

To update your IP address in the DNS, our servers need to determine the hostname you want to update (it's possible to set up several domains). To determine the hostname, we try the following steps until there is a match:

- `hostname` query string parameter, unless it is set to `YES` (this sometimes happens with dynDNS update clients).
- `host_id` query string parameter.
- The username as provided in the HTTP Basic Authorization header.
- The username as provided in the `username` query string parameter.
- After successful authentication (no matter how), the only hostname that is associated with your user account (if not ambiguous).

If we cannot determine a hostname to update, the API will return a `404 Not Found` status code. If the selected hostname is not eligible for dynamic updates, we will return `403 Forbidden`. This usually happens if you try updating a hostname that is not under the `dedyn.io` domain. If you are affected by this and would like to use another domain, please contact support.

8.1.3 Determine IP addresses

The last ingredient we need for a successful update of your DNS records is your IPv4 and/or IPv6 addresses, for storage in the `A` and `AAAA` records, respectively.

For IPv4, we will use the first IPv4 address it can find in the query string parameters `myip`, `myipv4`, `ip` (in this order). If none of them is set, it will use the IP that connected to the API, if a IPv4 connection was made. If no address is found or if an empty value was provided instead of an IP address, the `A` record will be deleted from the DNS.

For IPv6, the procedure is similar. We check `myipv6`, `ipv6`, `myip`, `ip` query string parameters (in this order) and the IP that was used to connect to the API for IPv6 addresses and use the first one found. If no address is found or an empty value provided instead, the `AAAA` record will be deleted.

8.2 Update Response

If successful, the server will return a response with status `200 OK` and `good` as the body (as per the `dyndns2` protocol specification). For error status codes, see above.

dynDNS updates are subject to rate limiting. For details, see [Rate Limits](#).

8.3 Examples

The examples below use `<your domain>.dedyn.io` as the domain which is to be updated and `<your authorization token>` as an API token affiliated with the respective account. (See *Manage Tokens* for details.) `<1.2.3.4>` is used as an example for an IPv4 Address, `<fd08::1234>` as a standin for an IPv6 address. Replace those (including the `<` and `>`) with your respective values.

Basic authentication with automatic IP detection (IPv4 or IPv6):

```
curl --user <your domain>.dedyn.io:<your authorization token> https://update.dedyn.io

curl https://update.dedyn.io/?hostname=<your domain>.dedyn.io \
  --header "Authorization: Token <your authorization token>"
```

Basic authentication with forced use of IPv6 (will remove IPv4 address from the DNS):

```
curl --user <your domain>.dedyn.io:<your authorization token> https://update6.dedyn.io

curl https://update6.dedyn.io/?hostname=<your domain>.dedyn.io \
  --header "Authorization: Token <your authorization token>"
```

Basic authentication with simultaneous update of IPv4 and IPv6:

```
curl --user <your domain>.dedyn.io:<your authorization token> \
  https://update.dedyn.io/?myipv4=1.2.3.4&myipv6=fd08::1234

curl https://update6.dedyn.io/?hostname=<your domain>.dedyn.io?myipv4=<1.2.3.4>&
  ↪myipv6=<fd08::1234> \
  --header "Authorization: Token <your authorization token>"
```

Endpoint Reference

The following table summarizes basic information about the deSEC API endpoints used for *managing users* and *tokens*.

Endpoint /api/v1...	Methods	Use case
.../auth/	POST	Register user account
.../auth/account/	GET	Retrieve user account information
.../auth/account/change-email/	POST	Request account email address change
.../auth/account/reset-password/	POST	Request password reset
.../auth/account/delete/	POST	Request account deletion
.../auth/login/	POST	Log in and request authentication token
.../auth/logout/	POST	Log out (= delete current token)
.../auth/tokens/	GET	Retrieve all current tokens
	POST	Create new token
.../auth/tokens/{id}/	GET	Retrieve token
	DELETE	Delete token
.../captcha/	POST	Obtain captcha
.../v/activate-account/{code}/	POST	Confirm email address for new account
.../v/reset-password/{code}/	POST	Confirm password reset
.../v/change-email/{code}/	POST	Confirm email address change
.../v/delete-account/{code}/	POST	Confirm account deletion

The following table summarizes basic information about the deSEC API endpoints used for *Domain Management* and *Retrieving and Manipulating DNS Information*.

Endpoint /api/v1/domains...	Methods	Use case
.../	GET	Retrieve all domains you own
	POST	Create a domain
.../{name}/	GET	Retrieve a specific domain
	PATCH	Modify a domain (deprecated)
	DELETE	Delete a domain
.../{name}/rrsets/	GET	Retrieve all RRsets from domain, filter by subname or type query parameter
	POST	Create one or more RRsets
	PATCH	Create, modify or delete one or more RRsets
	PUT	Create, modify or delete one or more RRsets
.../{name}/rrsets/@/{type}/		Access an RRset at the zone apex
.../{name}/rrsets/{subname}/ {type}/ .../{name}/rrsets/{subname}. .../{type}/	GET	Retrieve a specific RRset
	PATCH	Modify an RRset
	PUT	Replace an RRset
	DELETE	Delete an RRset

Rate Limits

The API implements rate limits to prevent brute force attacks on passwords, to ensure that the system load remains manageable, to avoid update rejections due to concurrent DNS updates on the same domain etc.

Rate limits apply per account. The following table summarizes the rate limits pertaining to various parts of the API. When several rates are given, all are enforced at the same time. For throttled requests, the server will respond with 429 Too Many Requests.

Rate limit name	Rate	Affected operations
account_management_act	3/min	Account activities with external effects (e.g. sending email)
account_management_pas	50/min	Account activities with internal effects (e.g. viewing account details, creating a token)
dyndns	1/min	dynDNS updates
dns_api_read	10/s 50/min	DNS read operations (e.g. fetching an RRset)
dns_api_write	6/s 50/min 200/h	DNS write operations (e.g. create a domain, change an RRset)
user	1000/day	Any activity of a) authenticated users, b) unauthenticated users (by IP)

API Versions and Lifecycle

To enable users to build reliable tools on top of the deSEC API, we maintain stable versions of the API for extended periods of time.

Each API version will advance through the API version lifecycle, starting from *unstable* and proceeding to *stable*, *deprecated*, and, finally, to *historical*.

Check out the [current status of the API versions](#) to make sure you are using the latest stable API whenever using our service in production.

Unstable API versions are currently under development and may change without prior notice, but we promise to keep an eye on users affected by incompatible changes.

For all **stable API versions**, we guarantee that

1. it will be maintained at least until the end of the given support period,
2. there will be no incompatible changes made to the interface, unless security vulnerabilities make such changes inevitable,
3. users will be warned before the end of the support period.

Deprecated API versions are going to be disabled in the future. Users will be notified via email and are encouraged to migrate to the next stable version as soon as possible. For this purpose, a migration advisory will be provided. After the support period is over, deprecated API versions may be disabled without further warning and transition to historical state.

Historical API versions are permanently disabled and cannot be used.

CHAPTER 12

Getting Help

If you need help beyond this documentation, please do not hesitate and shoot us an email at support@desec.io.

CHAPTER 13

About this document

To add to our documentation or fix a mistake, please submit a Pull Request at <https://github.com/desec-io/desec-stack>.