REST API Documentation

Version 1.7.1
2026-07-17

Dr. Michael Bauland, Knipp Medien und Kommunikation GmbH

Table of Contents

1. Introduction 3
    1.1. Accessing ironDNS® 3
2. REST API 3
    2.1. API Authentication 4
        2.1.1. Two Factor Authentication (2FA) 4
    2.2. Zone Management 5
        2.2.1. Create operation 5
        2.2.2. Initiate Fetch operations 10
        2.2.3. Delete operation 11
        2.2.4. Info operation 12
        2.2.5. Update operation 13
        2.2.6. List operation 15
        2.2.7. Key Rollover operation 15
        2.2.8. NotifyDSUpdate operation 16
        2.2.9. ImportDNSKEY operation 17
        2.2.10. ExportDNSKEYs operation 18
    2.3. Messages 19
        2.3.1. Poll operation 19
        2.3.2. Poll-ack operation 19
    2.4. Statistics 20
        2.4.1. CreateCollection operation 20
        2.4.2. DeleteCollection operation 22
        2.4.3. GetCollection operation 23
        2.4.4. UpdateCollection operation 23
        2.4.5. ListCollections operation 24
        2.4.6. ListAvailableStatistics operation 25

1. Introduction

The Internet's rising importance for enterprise-critical applications is posing rapidly increasing challenges to the operation of name servers for Internet addresses (domain names). In many cases, the evolved technical architectures have been insufficient for quite some time. But this is often recognized too late, namely when, e.g., DDoS attacks cause outages to Internet addresses.

Enterprises and domain registrars are often lacking the ability to deal with this issue properly. Moreover, it is advisable for them to focus on their core business and leave the operation of their DNS infrastructure to experts.

This gap can be addressed using the services provided by ironDNS®. It can supplement the customer's name server infrastructure, or even replace it completely.

The technology behind ironDNS® was developed from scratch by Knipp developers. The ironDNS® service represents the first global solution involving the ironDNS® engine. It has been deployed for several customers requiring mission-critical Internet infrastructure.

1.1. Accessing ironDNS®

There are three different ways to access and make use of the ironDNS® services:

- web Control Panel (https://manager.irondns.net/)
- SOAP API (see http://www.irondns.net/technology/provisioning/soap.faces)
- REST API (this documentation)

2. REST API

As summarized in the Introduction, the purpose of the system is to support the customer in creating, maintaining, and publishing DNS zones. Basically, there are two different ways for the customer to transfer his zone data to the ironDNS® zone manager:

- The zone data, i.e., its resource records, is submitted **literally** (either as a complete replacement of previous data, or as a so-called delta that describes removals and additions with regard to a specific previous version of a zone.
- The zone is set up with a primary **stealth name server** that feeds the zone data to the ironDNS® zone manager.

Subsequent sections of this document describe the API requests necessary to manage the customer's zones in both cases. In order to achieve a high degree of platform and programming language independence, the ironDNS® API is exposed as a **RESTful web service**, which offers a couple of simple operations for zone data provisioning. For each operation, a well-defined request is sent to the REST API's HTTPS endpoint (as an JSON document),

and a response (also in JSON) is returned to the client. All API endpoints and further information (like IP addresses of the fetchers) are published on the Control Panel at https://manager.irondns.net/v2/api.

All REST requests have to be send as an HTTP POST request using the header content-type application/json. For each operation the relative context path will be given. For example, if the REST API endpoint is https://api.irondns.net/rest, and the relative context path for zone create is zone/create, then the full URL to be used to create zones is https://api.irondns.net/rest/zone/create.

2.1. API Authentication

Each user of the ironDNS® REST API receives a set of credentials, consisting of a user name and a password (the same credentials can also be used for the Control Panel or the SOAP API), which have to be provided for each request sent to the REST interface (the API is stateless). Note: When a request is sent, the user name/password pair is part of the REST JSON data. Next to the actual payload a `credentials` JSON object has to be included. This contains a mandatory `userName` and a `password` entry, like this:

```
"credentials": {
       "userName": "myLogin",
       "password": "mySecret"
}
```

Please refer to https://www.json.org for more Information about JSON syntax.

To allow administrators to act on behalf of a certain user, there are two optional JSON fields `effCustomer` and `effCustomerSource` for all operations described in this document. If a user who is not an administrator specifies these attributes, the `effCustomer` has to match the ID of the customer issuing the request and the `effCustomerSource` has to match the user's source (or left empty for the user's own source).

2.1.1. Two Factor Authentication (2FA)

The REST API optionally supports 2FA for all requests. While 2FA for the Control Panel is handled via TOTP (time-based one-time passwords), both the REST and the SOAP API use client certificates as the second factor. 2FA can be activated on a customer level (forcing all users to use 2FA both in the Control Panel as well as for their API clients) or individually for each user account. To enable/disable 2FA for your customer, please contact the support team. The setting for individual users can be managed via Preferences → Miscellaneous in the Control Panel. In the section "Second factor for API access (SOAP/REST)." you can set your client certificate (either literally or by uploading a certificate file).

Once activated all REST requests need to include your client certificate. The certificate must be valid (i.e., not expired) and signed by any of the global standard certificate authorities.

2.2. Zone Management

This section describes the heart of the ironDNS® system, namely the management of zones. For this the basic operations to create, delete, and update a zone as well as obtain the zone's data via the info operation are defined. Additionally a list operation allows to get an overview of all zones. Finally the notifyDSUpdate operation is used in the DNSSEC context to inform ironDNS® about an update of the zone's DS record within its parent zone.

As an additional security feature ironDNS® introduces the two factor authentication for zone changes. This optional feature allows customers to lock important zones as a safeguard against unintentional or unauthorized updates. To enable the feature an e-mail address together with its public PGP key has to be provided to the ironDNS® support team (please contact support@ironDNS.net for details).

In order to use the zone locks within the REST interface the create, update, initiateFetch, delete, and notifyDSUpdate operations each have an optional `otpData` element in their request. This element can be used to obtain a new One Time Password (OTP) for the operation to be executed (by setting the `generateOtp` sub-element to true). In which case the operation is not executed, but an OTP is generated and sent via encrypted e-mail. Optionally the `clientId` sub-element can be used to set an arbitrary token which will also be included in the encrypted e-mail (making it easier to connect the e-mail with the intended operation). The result of such a request contains the `otpGenerated` attribute set to true informing the customer that the actual request has not been executed, but instead the OTP has been generated and sent in a PGP encrypted e-mail. After receiving the OTP the same operation has to be executed a second time. This time without the `generateOtp` sub-element and instead submitting the received OTP in the `otp` sub-element. An example of the workflow can be found in the section describing the delete operation (see 2.2.3).

2.2.1. Create operation

Relative path: zone/create

This operation creates a new zone object with an arbitrary ID and a static domain name. Once the zone has been created, neither its ID nor its domain name can be changed. For every zone one of the user's available products has to be selected. By omitting the product name in the create request, the user's default product will be used (the default product can be changed in the ironDNS® Control Panel). Furthermore, the list of target name servers has to be supplied. These are the name servers on which the zone will be deployed. For the zone's content there are two choices: either the literal content or the list of source name servers is given. Optionally the customer can decide for which events he wants to receive notification poll messages, how strict the zone validation should be carried out, and which ACL template should be used for the zone. ACL (= Access Control List) templates allow full zone transfers to certain IP addresses and can be managed via the ironDNS® Control Panel. Finally the user can assign an arbitrary comment to the zone which serves solely informational purposes and may help the user administrating his zones.

Next to the ironDNS® name servers it is also possible to use one or more secondary name servers operated by the user for which some ironDNS® name servers act as primary

(stealth) name servers. The setup for this scenario is very easy. The create operation takes a list of secondary name servers (IPs) as an optional argument and returns a list of primary stealth name servers. The primary stealth name servers cannot be chosen by the user but will be automatically selected. They will send notifies and allow zone transfers to the configured secondary name servers, but will not be publicly visible. This way it is possible, for example, to use the ironDNS® DNSSEC feature for easy zone signing while still operating one or more name servers in-house.

The create request contains the following allowed elements (wherein `content` and `src` are mutually exclusive and exactly one of them has to be defined):

`zoneid` the optional ID of the zone, used for referencing the zone in all other requests (if omitted an ID will be generated)

`product` the zone's product name; if omitted the user's default product will be used for the zone creation

`domainName` the domain name the zone is created for; cannot be modified

`notifications` specifying a list of notifications that should result in a poll message; possible notifications are `USER_DEPLOYMENT` (notification about a finished user-induced zone update), `SYSTEM_DEPLOYMENT` (notification about a finished system-induced zone update), `TRANSFER_FAILURE` (notification about a failed zone transfer from the master server), `PARENT_DS_UPDATE` (notification about the requirement to update the DS record at the parent zone, i.e., in most cases the registry), `GENERAL_ERROR` (notification about a general error not caused by ironDNS®), and `VALIDATION_ERROR` (notification about an error caused during zone validation; zones are not deployed to the name server in case of such errors).

`nameServers` specifying a list of strings, each of which represents one of the possible destination name servers

`aclTemplate` specifying one of the ACL templates to be used to define access to this zone; the available ACL templates can be managed via the ironDNS® Control Panel

`dnssec` DNSSEC specific configuration; this is a boolean attribute, which defines whether ironDNS® should sign the zone and generate DS records for the parent zone; setting it to false has the same effect as omitting the `dnssec` element

`dnssecProfileLabel` DNSSEC specific configuration; this is a string attribute, which defines what DNSSEC profile and therefore which signing algorithm ironDNS® should use. When omitting this string ironDNS® will use the default profile.

`zskData` an optional multi-line text element with the to be imported private ZSK DNSKEY in BIND's Digital Signature Trust (DST) text format.

`kskData` an optional multi-line text element with the to be imported private KSK DNSKEY in BIND's Digital Signature Trust (DST) text format.

`lockMode` the mode used to lock the zone after creation; if missing the default value of `NO_LOCK` is used which means that the zone is not locked; the other two possible values are `LOCKED_WITH_FETCH` and `LOCKED_NO_FETCH`; in both cases all further updates and the deletion of the zone have to be authorized by providing a One Time Password; the only difference is for zones having source name servers: the former lock still allows regular

updates of the zone content from its source name servers, while the latter mode also prohibits those automatic updates and each zone transfer will have to be initiated and authorized using an OTP; see also the general explanation in Section 2.2

`content` the literal content of the zone; this element takes the optional element `version`, which tags the zone content with a version token; the actual content is contained in an `rfc` element whose content is strongly interpreted according to the rules of RFC 1034/1035; it is possible to omit the SOA record and/or the NS records, these will then be generated automatically by the ironDNS® zone manager

`src` the definition of a list of self-run Master name servers or hidden Primary/Stealth name servers; this JSON array contains one or more objects defining a single primary name server; each of those takes a mandatory `ipAddresses` element containing one or more objects containing the source server's IPv4 or IPv6 addresses; each address may have an an optional `port` element to contact the source name server on a port other than the standard 53 port; optionally each JSON object in the `src` element may contain a `<comment>` tag for a user-defined arbitrary comment describing the server, a `priority` element (possible values are `LOW`, `MEDIUM` (default if omitted), and `HIGH`) and one `tsig` element describing the server's TSIG information supplied via the three elements `algorithm` for the TSIG key's algorithm (possible values are `HMAC_MD_5`, `HMAC_SHA_224`, `HMAC_SHA_256`, `HMAC_SHA_384`, and `HMAC_SHA_512`) and name for the TSIG key's name; the BASE64 encoded TSIG key is submitted as the tag's content

`comment` an optional comment (up to 64 characters) describing the zone or its usage; this value has no impact on the zone's operation

`promotionCode` an optional promotion code (16 characters long separated into four groups divided by an '-' e.g. 'AfxQ-d5bN-Vzv2-vzJp') used to create a zone without being charged for a defined amount of months.

`validationMode` optional element to specify one of the four validation modes (`STRONG`, `NORMAL`, `SOFT`, `OFF`); if no value is given the user's default setting is used (which can be changed via the ironDNS® Control Panel)

`secondaryServers` the definition of a list of one or more self-run secondary name servers; this JSON array contains one or more objects; each of those takes one or more objects containing the source server's IPv4 or IPv6 addresses; each address may have an an optional `port` element to contact the secondary name server on a port other than the standard 53 port; optionally each JSON object in the `src` element may contain a `comment` element for a user-defined arbitrary comment (up to 256 characters) describing the server, a boolean `notify` element to define whether notifies should be sent to the name server (default is true) and one `tsig` element describing the server's TSIG information supplied via the three elements `algorithm` for the TSIG key's algorithm (possible values are `HMAC_MD_5`, `HMAC_SHA_224`, `HMAC_SHA_256`, `HMAC_SHA_384`, and `HMAC_SHA_512`) and name for the TSIG key's name; the BASE64 encoded TSIG key is submitted as the tag's content

`otpData` the data used to create a locked zone and generate and enter the necessary One Time Password; see also the general explanation in Section 2.2

Example (with literal content)

Create a zone for the domain example.com with the ID testZone1 (used for referring to the zone during delete, update, and info requests). The user requests notifications whenever the zone has been deployed through user intervention. The zone shall be deployed on the two name servers `ns1.irondns.net` and `ns2.irondns.net`. Since neither a SOA nor NS records are submitted via the literal content element, those records will be generated automatically.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "product": "Basic 1",
  "domainName": "example.com",
  "notifications": ["USER_DEPLOYMENT"],
  "nameServers": ["ns1.irondns.net", "ns2.irondns.net"],
  "aclTemplate": "my test template",
  "dnssec": false,
  "lockMode": "NO_LOCK",
  "content": {
    "rfc": "www  3600 IN A    193.232.12.23\n  AAAA 234:231::\nsub 900 IN A
127.0.0.123\nsub2 1800 IN A 127.0.0.124",
    "version": "myVersionID"
  },
  "comment": "My example domain.",
  "validationMode": "STRONG"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false,
  "zoneid": "testZone1"
}
```

Example (with source name servers)

Create a zone for the domain example.com with the ID `testZone2`. The user requests notifications whenever the zone has been deployed (no matter how the deployment has been initiated), whenever a transfer failure occurred (e.g., source name server is not reachable), in case of validation errors, and in case of other not specified errors. The zone shall be deployed on the two name servers `ns1.irondns.net` and `ns2.irondns.net` and at first the name server with address 127.0.0.2 (IPv4) or ::2 (IPv6) will be queried (since the server has the default priority medium) using the given TSIG data. If this fails, the second server with low priority is queried (without TSIG validation). The second server uses the port number 1053 instead of the standard 53 port. Only if the zone cannot be transferred from both servers, an error message is generated and added to the user's poll message queue.

Additionally a secondary name server (without TSIG) with address 192.168.0.1 shall be used. The response tells what hidden primary name servers will provide the zone content

for transfer and will send notifies (in this case `ns1.hidden.irondns.net` and `ns2.hidden.irondns.net`).

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone2",
  "product": "Basic 1",
  "domainName": "example2.com",
  "notifications": ["TRANSFER_FAILURE",
    "USER_DEPLOYMENT",
    "SYSTEM_DEPLOYMENT",
    "GENERAL_ERROR",
    "VALIDATION_ERROR"],
  "nameServers": ["ns1.irondns.net", "ns2.irondns.net"],
  "aclTemplate": "my test template",
  "src": [
    {
      "priority": "LOW",
      "comment": "Secondary Name Server (US)",
      "ipAddresses": [{
          "ipAddress": "127.0.0.1",
          "port": 1053
        }, {
          "ipAddress": "::1",
          "port": 1053
        }]
    },
    {
      "comment": "Primary Name Server (Head Quarters)",
      "ipAddresses": [{
          "ipAddress": "127.0.0.2"
        }, {
          "ipAddress": "::2"
        }],
      "tsig": {
        "value": "eW91IHNOaW5rIGxpa2UgbW9ua2V5IGJ1dHQ=",
        "algorithm": "HMAC_SHA_1",
        "name": "a.b.c.d"
      }
    }
  ],
  "comment": "My other example domain.",
  "validationMode": "NORMAL",
  "secondaryServers": [
    {
      "comment": "Secondary Server (CAN)",
      "ipAddresses": [{
          "ipAddress": "192.168.0.1"
        }],
      "notify": true
    }
  ]
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false,
  "hiddenPrimaryServers": ["ns1.hidden.irondns.net",
"ns2.hidden.irondns.net"],
  "zoneod": "testZone2"
}
```

2.2.2. Initiate Fetch operations

Relative path: zone/fetch

This operation initiates a zone fetch operation. It can only be used for zones having source name servers. Zones with literal content will result in an error message. As a result of the successful execution of this operation the zone will immediately be scheduled for fetching, i.e., the ironDNS® Fetcher will contact the zone's source name server(s) and check if a new version is available. If so, the zone will be updated with the new version.

The request contains the mandatory element `zoneid` which specifies the ID of the zone object to be fetched. Further allowed elements in the request are the `force` flag (to determine whether a fetch should be initiated even if the serial has not changed) and is the optional `otpData`. It has to be used to generate and submit the One Time Password needed to fetch locked zones, see also the general explanation in Section 2.2.

Example

Initiate fetching of the (unlocked) zone with ID testZone1 even if the serial has not changed.

Request

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "force": "true"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

2.2.3. Delete operation

Relative path: zone/delete

This operation deletes a zone object.

The request contains the mandatory element `zoneid` which specifies the ID of the zone object to be deleted. The only further allowed element in the request is the optional `otpData`. It has to be used to generate and submit the One Time Password needed to delete locked zones, see also the general explanation in Section 2.2.

Example (delete a non-locked zone)

Delete the zone with the ID testZone1.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

Example (delete a locked zone)

Delete the locked zone with the ID testZone2. The example consists of two separate requests. At first the necessary One Time Password (OTP) needs to be generated in Request 1. The OTP is then send in an encrypted e-mail and has to be submitted in a second request.

Request 1 (generate OTP):

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "otpData": {
    "generateOtp": true,
    "clientId": "deleteTestZone2"
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": true
}
```

Request 2 (delete zone with OTP):

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "otpData": {
    "otp": "123456"
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

2.2.4. Info operation

Relative path: zone/info

This operation retrieves the data associated with the zone.

The request contains the mandatory element `zoneid` specifying the ID of the zone object to retrieve. Optionally the element `include` may be used to selectively retrieve partly information about the zone (if omitted all information will be retrieved). Possible values (space separated) are:

`CONFIG` retrieve the zone's configuration, i.e., the product name, domain name, comment, target and source name servers, secondary and hidden primary name servers, the validation mode, whether the zone is locked, and the used ACL template (if any)

`CONTENT_RECEIVED` retrieve the zone's content, as received from the user via literal data or from the master server

`CONTENT_PUBLISHED` retrieve the zone's content, as published on the name servers (includes auto-generated SOA and NS records, as well as DNSSEC enhancements if enabled)

`DEPLOYMENT` retrieve deployment (version/serial number) information for the zone

`DNSSEC` retrieve DNSSEC related information for the zone; in particular this includes the DNS key and respective DS records that need to be submitted to and deployed in the parent zone in order to establish a chain of trust

Example

Obtain configuration and deployment information for the zone with ID testZone3.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone3",
  "include": ["CONFIG", "DEPLOYMENT"]
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "product": "Basic 1",
  "domainName": "example.com",
  "notifications": ["USER_DEPLOYMENT"],
  "nameServers": ["ns2.irondns.net", "ns1.irondns.net"],
  "aclTemplate": "my test template",
  "deployment": {
    "oldestVersion": "myVersionID",
    "newestVersion": "myVersionID",
    "oldestSerial": "1",
    "newestSerial": "1"
  },
  "comment": "My example domain.",
  "validationMode": "STRONG",
  "secondaryServers": [{
      "ipAddresses": [{
          "ipAddress": "192.168.0.1"
        }],
      "notify": true
    }],
  "hiddenPrimaryServers": ["ns1.hidden.irondns.net", "ns2.hidden.irondns.net"]
}
```

2.2.5. Update operation

Relative path: zone/update

This operation updates the data for the given zone object. Similarly to the Create operation the zone's configuration and/or data can be changed. Apart from the zone's ID and domain name every data supplied in the create request can be updated. Additionally it is possible to update the literal content (if it exists) via the delta mechanism, i.e., provide resource records that should be removed from or resource records that should be added to the existing zone content.

The request contains the mandatory element `zoneid`. Allowed elements in the request are `product`, `notifications`, `nameServers`, `aclTemplate`, `dnssec`, `lockMode`, `content`, `src` (wherein `content` and `src` are mutually exclusive and exactly one of them has to be defined), `comment`, `zskData`, `kskData`, `validationMode`, the `secondaryServers`, and the

`otpData`. Since all elements except `content` are the same as in the Create operation, we will only describe the `content` in more detail.

As in the create request, the `content` element takes the optional element `version`, which tags the zone content with a version token. Within the `content` element there is the choice between the two elements `replace` and `delta`. The former takes the new content in an `rfc` element as defined in the create request. The latter element has the optional element `fromVersion`, which can be used to ensure the current zone is the one expected. If the from version is submitted, it has to be the same as the zone's current version token, otherwise an error is reported. Furthermore, the `delta` element contains the two optional elements `add` and `rem`, which both take an `rfc` element as defined in the create request. Any records defined in the `rem` element will be removed from the zone (an error will be generated if such a record does not exist) and any record from the `add` element will be added to the existing records.

If the zone is setup with secondary name servers for the first time the response contains the hidden primary name servers to be used for the secondary name servers. In all other cases the response is empty. If a zone with secondary name servers has its secondary name servers updated the assigned hidden primary name servers will always stay the same.

Example

Update the zone with ID `testZone1`. Change the product to "Basic 2", leave notification settings, ACL template, destination name server settings, secondary name server settings, and the validation unchanged, but update the zone's content, by removing the record `sub2 1800 IN A 127.0.0.124` and adding `sub5 1800 IN A 127.0.0.125`.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "product": "Basic 1",
  "content": {
    "version": "myVersionNewID",
    "delta": {
      "fromVersion": "myVersionID",
      "add": {
        "rfc": "sub5 1800 IN A 127.0.0.125"
      },
      "rem": {
        "rfc": "sub2 1800 IN A 127.0.0.124"
      }
    }
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

2.2.6. List operation

Relative path: zone/list

This operation lists all zones (zone ID and domain name) associated with a user.

There are no particular elements for this kind of request.

Example

List all zones associated with the current user.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "zones": [{
      "zoneid": "testZone1",
      "domainname": "example.com"
    },
    {
      "zoneid": "testZone2",
      "domainname": "example2.com"
    }
  ]
}
```

2.2.7. Key Rollover operation

Relative path: zone/key-rollover

This operation initiates a key rollover for the zone with the given ID. If `zsk` is set to true a key rollover for the zone signing key will be initiated. Otherwise a key signing key rollover will be initiated.

Example

Initiate a key signing key rollover.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone1",
  "zsk": false
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }]
}
```

2.2.8. NotifyDSUpdate operation

Relative path: zone/notifyDSUpdate

This operation tells ironDNS® that the DS record has been updated at the parent zone. When ironDNS® manages the signing of your zone it will generate a DNS key and its respective DS record. Whenever new DNSSEC data is generated you will receive a corresponding poll message and can obtain the data via the Info command. After you have transmitted this data to the zone's parent (and it has been propagated to all respective name servers) you will need to send this REST request to inform ironDNS® about the successful update.

The request contains the mandatory element `zoneid` which specifies the ID of the zone object whose DS record has been updated. The only further allowed element in the request is the optional `otpData`. It has to be used to generate and submit the One Time Password needed to update locked zones, see also the general explanation in Section 2.2.

Example

Inform the zone signing component of ironDNS® about the fact that the DS record for the zone with ID testZone2 has been updated at the parent zone.

Request

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone2"
}
```

Response

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

2.2.9. ImportDNSKEY operation

Relative path: zone/importDNSKEY

This operation allows to import your own private DNSKEY (either KSK or ZSK) into ironDNS®. When importing such a key, it is automatically scheduled for use, i.e., a key rollover is started as a consequence.

The request contains the mandatory element `zoneid` which specifies the ID of the zone object whose DNSKEY should be imported. Further mandatory elements are

`zsk` a boolean flag, which defines whether the imported key should be used as a ZSK or KSK

`keyData` a multi-line text element with the to be imported private DNSKEY in BIND's Digital Signature Trust (DST) text format.

The only further allowed element in the request is the optional `otpData`. It has to be used to generate and submit the One Time Password needed to update locked zones, see also the general explanation in Section 2.2.

Example

Import a KSK for the zone with ID testZone2.

Request

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone2",
  "zsk": "false",
  "keyData": "Private-key-format: v1.3\n Algorithm: 13 (ECDSAP256SHA256)\n
PrivateKey: ZYcYhR5f98vI1+BFGKLIrarZrqxJM4mRy9tvwntdYoo=\n Created:
20000101000000\n Publish: 20000101000000\n Activate: 20000101000000"
}
```

Response

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false
}
```

2.2.10. ExportDNSKEYs operation

Relative path: zone/exportDNSKEYs

This operation allows to export your own private DNSKEYs (either KSKs or ZSKs) from ironDNS®. Unless the zone is currently in a DNSKEY rollover phase, there will be one KSK or one ZSK exported. During a rollover phase, multiple keys will be exported.

The request contains the mandatory elements `zoneid` which specifies the ID of the zone object whose DNSKEYs should be exported and the flag `zsk` which defines whether the ZSK or KSK should be exported. The only further allowed element in the request is the optional `otpData`. It has to be used to generate and submit the One Time Password needed to update locked zones, see also the general explanation in Section 2.2.

Example

Export a ZSK for the zone with ID testZone2 having a ZSK rollover and therefore two current keys.

Request

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "zoneid": "testZone2",
  "zsk": "true"
}
```

Response

```
{
  "result": [{
      "code": "success"
    }],
  "otpGenerated": false,
  ["Private-key-format: v1.3\n Algorithm: 13 (ECDSAP256SHA256)\n PrivateKey:
B8fNHNrxVoWbbVLQsTqqnI3WJ9A51P5oOAhxH45xxqA=\n",
"Private-key-format: v1.3\nAlgorithm: 13 (ECDSAP256SHA256)\nPrivateKey:
z9HBbzfBKOyyBum4CsWoyGctEyHA693/78cHQabTfGc=\n"]
}
```

2.3. Messages

2.3.1. Poll operation

Relative path: message/poll

This operation retrieves the user's oldest poll message. In order to retrieve the next poll message, the oldest one has to be acknowledged first. There are five different poll message types, which correspond to the five notification types defined in the Create operation in Section 2.2.1.). The notification types, therefore the name of the element in the JSON, are `userDeployment`, `systemDeployment`, `transferFailure`, `parentDsUpdate`, and `generalError`.

There are no particular elements for this kind of request.

Example

Retrieve the oldest (not acknowledged) poll message.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "msg": {
    "systemDeployment": {
      "message": "A user-initiated deployment has been completed: the zone
example.com with serial number 1 has been successfully deployed on all name
servers.",
      "zoneid": "example.com"
    },
    "msgid": "1",
    "date": "2020-10-17T08:39:08.413+0000"
  }
}
```

2.3.2. Poll-ack operation

Relative path: message/pollAck

This operation acknowledges a certain poll message, i.e., deletes the poll message. This does not have to be the oldest poll message, but can be an arbitrary one. In order to read the second oldest poll message, the oldest has to be acknowledged first.

The request contains the mandatory element `msgid` to define, which poll message should be acknowledged.

Example

Acknowledge/delete the poll message with ID 1.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "msgid": "1"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }]
}
```

2.4. Statistics

The ironDNS® statistics component allows the user to define different ways how the DNS traffic should be gathered and aggregated. A combination of statistic aggregation parameters is called Collection. Using the REST operations createCollection and updateCollection it is possible to define such Collections and start the statistic gathering process in the background. Reports will then be generated automatically and with the REST operation listAvailableStatistics a filter can be used to find the required statistic reports.

2.4.1. CreateCollection operation

Relative path: stats/create

This operation creates a new statistic Collection. The Collection has six parameters that define how and which data should be gathered. It is possible to either collect all data for certain (explicitly specified) zones only or to collect the data for all zones in the user's account. Furthermore, the query types and the report intervals can be set.

The request contains the mandatory element `name`. The name of the Collection must be unique and is used as an identifier for the following operations. It is also needed for retrieving the reports generated by the Collection. The allowed elements of the request are:

`zoneids` the list of zones (referenced by their ID) that should be included in the generated reports; if omitted the reports will contain the data for the DNS requests related to any of the user's active zones

`intervals` specifying the list of intervals that should be used for the report generation; at least one interval has to be given; if more than one interval is given, reports for each interval will be generated; possible values are `ONE_MINUTE` (1 min), `FIVE_MINUTES` (5 min), `FIFTEEN_MINUTES` (15 min), `ONE_HOUR` (1 h), `SIX_HOURS` (6 h), `ONE_DAY` (1 d), `ONE_WEEK` (1 wk), and `ONE_MONTH` (1 mo); each report will combine one or more files having the defined interval (e.g., specifying the one minute interval will generate a report every hour containing 60 files each having the data of one minute).

`recordTypes` specifying the DNS queries that should be counted; possible values are `A` (A records, IPv4 address), `AAAA` (AAAA records, IPv6 address), `MX` (MX records, e-mail), `NS` (NS records, name server), `NAPTR` (NAPTR records, name authority pointer), `TXT` (TXT records, text), `DNSSEC` (DNSSEC-related records, i.e., RRSIG, NSEC, NSEC3, NSEC3-PARAM, DS, DNSKEY), `CNAME` (Canonical Name records, redirect), and `OTHER` (all other queries); note that if `OTHER` is selected, all not selected record types will be counted as Other records (e.g., if you only select `A` and `OTHER`, a query for the AAAA record will be counted towards the Other records); at least one record type or `OTHER` needs to be specified.

`reportType` specifying the type of the report to be generated. Possible values are:

- `GENERAL`: this generates the usual report depending on all input parameters and grouping the data by source IP address, record type and domain name; for each record type / domain name combination only the top source IP addresses are listed individually, while the others are combined;
- `NON_EXISTING`: this is a special report type for registries and can only be used with a single zone; the report will then contain the top queries to domain names which do not exist within the zone
- `ICANN_REPORT`: this is a special report for registry zones for which an ICANN report according to the new gTLD policies has to be created; the report may only be used with an interval of one month and a single zone; it sums up all queries received and answered for that zone during that interval grouped by their protocol type (UDP vs. TCP)

`reportCutOff` optional value specifying maximum number of IP addresses (for the General report) or domain names (for the `NON_EXISTING` report) after which all further lines are combined. If missing, a suitable default value is used.

`ignoreSourceIPs` specifying whether the source IP addresses are ignored and only the domain names and record types are considered. This value is optional, if missing it is set to `false`. It is only used for reports of type `GENERAL`.

Example

Start the DNS query statistic logging for the two zones having the ID myZone1 and myZone2, respectively. The aggregation interval should be one minute and one day and the DNS queries should be divided into queries for A records and all other queries. The reports can be accessed using the Collection name testCollection.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "name": "testCollection",
  "zoneids": ["myZone1", "myZone2"],
  "intervals": ["ONE_MINUTE", "ONE_DAY"],
  "recordTypes": ["A", "OTHER"],
  "reportType": "GENERAL",
  "reportCutOff": 50,
  "ignoreSourceIPs": false
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }]
}
```

2.4.2. DeleteCollection operation

Relative path: stats/delete

This operation deletes a Collection. The aggregation of DNS queries will be stopped immediately. All previously generated reports will also be deleted. Therefore it should be ensured that all still needed data is downloaded before the execution of this operation. The operation is irreversible. Note that the deletion may take a short while. During this time the data Collection cannot be accessed anymore, but neither can its name be used for a new Collection. After the deletion is completed, the Collection name is free again and can be used in the create request to create another Collection.

The request contains the mandatory element `name` specifying the name of the Collection to be deleted.

Example

Delete the Collection with the name testCollection.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "name": "testCollection"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }]
}
```

2.4.3. GetCollection operation

Relative path: stats/info

This operation retrieves the parameters associated with the Collection.

The request contains the mandatory element `name` specifying the name of the Collection to retrieve. The response then contains the same elements that can be specified in the create request.

Example

Obtain configuration parameters for the Collection with name testCollection.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "name": "testCollection"
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "name": "testCollection",
  "zoneids": ["myZone1", "myZone2"],
  "intervals": ["ONE_MINUTE", "ONE_DAY"],
  "recordTypes": ["A", "OTHER"],
  "reportType": "GENERAL",
  "reportCutOff": 50,
  "ignoreSourceIPs": false
}
```

2.4.4. UpdateCollection operation

Relative path: stats/update

This operation updates the parameters for the given Collection. Similarly to the create operation the Collection's configuration can be changed. Apart from the Collection's name and the report type, every data supplied in the create request can be updated. The name element will be used to identify the Collection to be updated. Note that in order to change the type of the report it is necessary to delete the old Collection and create a new one.

The request contains the mandatory element `name`. Allowed elements in the request are `zones`, `intervals`, `recordTypes`, and the `reportCutOff`. Since all these elements have the same meaning and possible values as in the create request, their description can be looked up at the create operation. Note, that the `reportType` and the `ignoreSourceIPs` values can not be changed for an existing collection.

Example

Update the DNS query statistic logging with the name testCollection. The statistics for all zones in the user's account should be gathered and summed up to daily reports. The reports should only contain A and NS records.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  },
  "name": "testCollection",
  "zoneids": ["myZone1", "myZone2"],
  "intervals": ["ONE_DAY"],
  "recordTypes": ["A", "NS"],
  "reportCutOff": 60
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }]
}
```

2.4.5. ListCollections operation

Relative path: stats/list

This operation lists all active Collections associated with a user.

There are no particular elements or attributes for this kind of request. The response contains a list of names. The actual parameters of the Collections have to be obtained using the `getCollection` operation.

Example

List all Collections associated with the current user.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  }
}
```

Response:

```
{
  "result": [{
      "code": "success"
    }],
  "names": ["testCollection"]
}
```

2.4.6. ListAvailableStatistics operation

Relative path: stats/listStatistics

This operation allows to get a list of all available statistic reports. The request has several filter parameters to narrow down the result list of reports. The response contains a list of links which can be used to download the report files. In order to access the link, the user needs to be logged in to the ironDNS® Control Panel. The credentials are the same as the ones for this SOAP interface.

The allowed elements of the request are:

`name` the name of the Collection that generated the reports; if omitted all available reports within the specified parameter range are returned

`fileFormat` the format of the report data files; currently only `TSVZIP` is a supported file format; this format describes a ZIP file that contains a content.xml file containing the metadata for the downloaded file (i.e., it lists the data files and their parameters including the header description of the TSV files) and the data files; each data file is in TSV (tab separated values) format containing the aggregated statistic samples

`from` only reports containing data after the given date will be listed; if omitted there is no restriction

`to` only reports containing data before the given date will be listed; if omitted there is no restriction

`minAggrInterval` this duration value defines the minimum aggregation interval for the returned reports; if omitted there is no restriction

`maxAggrInterva` this duration value defines the maximum aggregation interval for the returned reports; if omitted there is no restriction

Example

List the available reports generated by the Collection testCollection having the file format TSVZIP (this could be omitted as there are no other file formats for the time being). The result should only contain reports that contain data from January 31, 2012, the minimum report interval should be one minute and the maximum report interval should be 30 minutes.

Request:

```
{
  "credentials": {
    "userName": "User-Name",
    "password": "xxxxxx"
  "name": "testCollection",
  "fileFormat": "TSVZIP",
  "from": "2012-01-31T00:00:00.000+01:00",
  "to": "2012-02-01T00:00:00.000+01:00",
  "minAggrInterval": "PT1M",
  "maxAggrInterval": "PT30M"
}
```

Response:


```
{
  "result": [{
      "code": "success"
    }],
  "statistics": [{
      "zoneid": "https://manager.irondns.net/report/download/testCollection-
20120131-0000-60.zip",
      "name": "testCollection",
      "fileFormat": "TSVZIP",
      "collectionStart": "2012-01-31T00:00:00.000+01:00",
      "collectionEnd": "2012-01-31T01:00:00.000+01:00",
      "aggrInterval": "P0Y0M0DT0H1M0.000S"
    },
    {
      "zoneid": "https://manager.irondns.net/report/download/testCollection-
20120131-0100-60.zip",
      "name": "testCollection",
      "fileFormat": "TSVZIP",
      "collectionStart": "2012-01-31T01:00:00.000+01:00",
      "collectionEnd": "2012-01-31T02:00:00.000+01:00",
      "aggrInterval": "P0Y0M0DT0H1M0.000S"
    },
    {
      "zoneid": "https://manager.irondns.net/report/download/testCollection-
20120131-0200-60.zip",
      "name": "testCollection",
      "fileFormat": "TSVZIP",
      "collectionStart": "2012-01-31T02:00:00.000+01:00",
      "collectionEnd": "2012-01-31T03:00:00.000+01:00",
      "aggrInterval": "P0Y0M0DT0H1M0.000S"
    }
  ]
}
```