Registration Protocols Extensions
Internet-Draft
Intended status: Informational
Expires: July 22, 2018

A. Blinn
R. Carney
GoDaddy Inc.

January 18, 2018

Domain Connect API - Communications between DNS Provider and Services
draft-carney-regext-domainconnect-03

Abstract

This document provides information related to the Domain Connect API that was built to support communications between DNS Providers and Service Providers (hosting, social, email, hardware, etc.).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 22, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Definitions 3
4. The API 4
4.1. The Synchronous Flow 5
4.2. The Asynchorous Flow 6
4.3. DNS Provider Initiated Flows 6
4.4. DNS Provider Discovery 7
4.5. Domain Connect Details 8
4.5.1. Synchronous Flow 9
4.5.1.1. Query Supported Template 9
4.5.1.2. Apply Template 9
4.5.1.3. Security Considerations 10
4.5.2. Asynchronous Flow: OAuth 12
4.5.2.1. Getting an Authorization Code 12
4.5.2.2. Requesting an Access Token 13
4.5.2.3. Making Requests with Access Tokens 14
4.5.2.4. Apply Template to Domain 15
4.5.2.5. Revert Template 16
4.5.2.6. Revoke Access 17
4.6. Domain Connect Objects and Templates 17
4.7. Operational and Implementation Considerations 19
5. IANA Considerations 23
5.1. XML Namespace 23
6. Acknowledgements 23
7. Change History 23
7.1. Change from 02 to 03 23
7.2. Change from 01 to 02 23
7.3. Change from 00 to 01 23
8. Normative References 23
Authors' Addresses 24

1. Introduction

Configuring a service at a Service Provider to work with a domain has historically been a complex task that is difficult for users.

Typically, a customer would try to configure their service by entering their domain name with the Service Provider. The Service Provider then used a number of techniques with mixed reliability to discover the DNS Provider. This might include DNS queries for

nameservers, queries to whois, and mapping tables to determine the registrar or the company providing DNS.

Once the Service Provider discovered the DNS Provider, they typically gave the customer instructions for proper configuration of DNS. This

might include help text, screen shots, or even links to the appropriate tools.

Discovery of the DNS Provider in this manner is unreliable, and providing instructions to users would present a number of technologies (DNS record types, TTLs, Hostnames, etc.) and processes they didn't understand. These instructions authored by the Service Provider often quickly become out of date, further confusing the issue for users.

The goal of this specificatoin is to create a system where Service Providers can easily enable their applications/services to work with the domain names of their customers. This includes both discovery of the DNS Provider and subsequent modification of DNS.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented in order to develop a conforming implementation.

3. Definitions

The following definitions are used in this document:

- Service Providers - refers to entities that provide products and services attached to domain names. Examples include web hosting providers (such as Wix or SquareSpace), email Service Providers (such as Microsoft or Google) and potentially even hardware manufacturers with DNS-enabled devices including home routers or automation controls (such as Linksys, Nest, and Philips).

- o DNS Providers - refers to entities that provide DNS services such as registrars (e.g. GoDaddy, 1and1) or standalone DNS services (e.g. CloudFlare).
- o Customer/User - refers to the end-user of these services.
- o Templates/Service Templates - refers to a file that describes a set of changes to DNS and domain functionality to enable a specific service.
- o Root Domain refers to a registered domain (e.g. example.com or example.co.uk) or a delegated zone in DNS.
- o Sub Domain refers to a sub-domain of a root domain (e.g. sub.example.com or sub.example.co.uk).

4. The API

The system will be implemented using simple web based interactions and standard authentication protocols. The creation and modification of DNS settings will be done through the application of templates instead of direct manipulation of individual DNS records.

Templates are core to this proposal as they describe a service owned by a Service Provider, and contain all of the information necessary to to enable and operate/maintain a service.

The individual records may be identified by a group identifier. This allows for the application of templates in different stages. For example, an email provider might first set a TXT record to verify the domain, and later set an MX record to configure email delivery. While done separately, both changes are fundamentally part of the same service.

It is important that templates be constrained to an individual service, as later removal of a template would remove all associated records.

Templates can also contain variable portions, as often values of data in the template change based on the implementation and/or user of the Service Provider (e.g. the IP address of a service, a customer id, etc).

Configuration and onboarding of templates between the DNS Provider and the Service Provider is seen as a manual process. The template

is defined by the Service Provider and given to the DNS Provider. Future versions of this specification may allow for an independent repository of templates. For now, the templates are all published at http://domainconnect.org.

By basing the protocol on templates instead of DNS Records, several advantages are achieved. The DNS Provider has very explicit knowledge and control on the settings being changed to enable a service. The system is also more secure as templates are tightly controlled and contained.

To attach a domain name to a service provided by a Service Provider, the customer would first enter their domain name.

Instead of relying on examination of the nameservers and mapping these to DNS Providers, DNS Provider discovery would be handled through simple records in DNS and an API. The Service Provider can query for a specific record in the zone to determine a REST endpoint to initiate the protocol. A Domain Connect compliant DNS Provider

would return information about that domain and how to configure it using Domain Connect.

For the application of the changes to DNS, there are two use cases. The first is a synchronous web flow, and the second is an asynchronous flow using OAuth and an API.

It should be noted that a DNS Provider may choose to only implement one of the flows. As a matter of practice many Service Providers are based on the synchronous flow, with only a handful of them based on the asynchronous OAuth flow. So, many DNS providers may opt to only implement the synchronous flow.

It should also be noted that individual services may work with the synchronous flow only, the asynchronous flow only, or with both.

4.1. The Synchronous Flow

This flow is tailored for the Service Provider that requires a one-time synchronous change to DNS.

The user would first enter their domain name at the Service Provider

website.

After the Service Provider determines the DNS Provider, the Service Provider might display a link to the user indicating that they can "Connect their Domain" to the service.

After clicking the link, the user is directed to a browser window on the DNS Provider's site. This is typically in another tab or in a new browser window, but can also be in place navigation with a return url. This link would pass the domain name being modified, the service provider and template being enabled, and any additional parameters needed to configure the service.

Once at the DNS Provider site, the user would be asked to authenticate if necessary.

After authenticating at the DNS Provider, the DNS Provider would verify the domain name is owned by the user. The DNS Provider would also verify other parameters passed in are valid and would prompt the user to give consent for making the change to DNS. The DNS Provider could also warn the user of services that would be disabled by applying this change to DNS.

Assuming the user grants this consent, the DNS changes would be applied. Upon successful application of the DNS changes, the popup

window or tab would be closed. If in place the user would be redirected back to the service provider.

4.2. The Asynchorous Flow

The asynchronous OAuth flow is tailored for the Service Provider that wishes to make changes to DNS asynchronously with respect to the user interaction, or wishes to make multiple or additional changes to DNS over time.

The OAuth based authentication and authorization flow begins similarly to the web based synchronous flow. The Service Provider determines the DNS Provider and links to the consent dialog at the DNS Provider where the user signs in, the ownership of the domain is verified, and the consent is granted.

However, instead of applying the DNS changes on user consent, OAuth access is granted to the Service Provider. An OAuth access code is generated and handed back to the Service Provider. The Service Provider then requests an access (bearer) token.

The permission granted in the OAuth token is a right for the Service Provider to apply a template to the specific domain owned by a specific user.

The Service Provider would later call an API that applies this template to the domain, including any necessary parameters along with the access token(s). As in all OAuth flows, access can be revoked by the user at any time. This would be done on the DNS Provider's user experience.

If the OAuth flow is used, once a Service Provider has an OAuth token the Domain Connect API becomes available for use. The Domain Connect API is a simple REST service.

This REST service allows the application or removal of the changes in the template on a domain name. The domain name, user, and template must be authorized through the OAuth token and corresponding access token.

Additional parameters are expected to be passed as name/value pairs on the query string of each API call.

4.3. DNS Provider Initiated Flows

A DNS Provider may with to expose interesting services that the user could attach to their domain. An example would be suggesting to a

user that they might want to connect their domain to a partner Service Provider.

If the template for the service is static, it is possible to simply apply the template.

However, often the template has some dynamic elements. For this scenario, the DNS Provider need simply call a URL at the Service

Provider. The Service Provider can then sign the user in, collect any necessary information, and call the normal web-based flows described above.

4.4. DNS Provider Discovery

In order to facilitate discovery of the DNS Provider from a domain name, a domain will contain a record in DNS.

This record will be a simple TXT record containing a URL used as a prefix for calling a discovery API. This record will be named domainconnect.

An example of this record might contain:

domainconnect.godaddy.com

As a practical matter of implementation, the DNS Provider need not contain a copy of this data in each and every zone. Instead, the DNS Provider needs simply to respond to the DNS query for the domainconnect TXT record with the appropriate data.

How this is implemented is up to the DNS Provider.

For example, the DNS Provider may not store the data inside a TXT record for the domain, opting instead to put a CNAME in the zone and have the TXT record in the target of the CNAME.

Once the URL prefix is discovered, it can be used by the Service Provider to determine the additional settings for using Domain Connect on this domain at the DNS Provider. This is done by calling a REST API.

```
GET
https://{domainconnect}/v2/{domain}/settings
```

This will return a JSON structure containing the settings to use for Domain Connect on the domain name (passed in on the path) at the DNS Provider. This JSON structure will contain the following fields:

- providerName: The name of the DNS Provider suitable for display on

the Service Provider UX.

- urlSyncUX: The URL Prefix for linking to the UX elements of Domain Connect for the synchronous flow at the DNS Provider.
- urlAsyncUX: The URL Prefix for linking to the UX elements of Domain Connect for the asynchronous flow at the DNS Provider
- urlAPI: This is the URL Prefix for the REST API.
- width: This is the width of the popup window for granting consent when navigated in a popup. Default value is 750px.
- height: This is the height of the popup window for granting consent when navigated in a popup. Default value is 750px.

As an example, the JSON returned by this call might contain.

```
{
"providerName": "GoDaddy",
"urlSyncUX": "https://domainconnect.godaddy.com",
"urlAsyncUX": "https://domainconnect.godaddy.com",
"urlAPI" : "https://api.domainconnect.godaddy.com",
"width" : 750,
"height" : 750
}
```

If the DNS Provider is not implementing the synchronous flow, the urlSyncUX is not required. Similarly, if the DNS Provider is not implementing the asynchronous flow the urlAsyncUX is not required.

4.5. Domain Connect Details

Domain Connect contains endpoints in the form of URLs.

The first set of endpoints are for the UX that the Service Provider links to. These are for the UX which includes the web-based flow where the user clicks on the link, and the OAuth flow where the user clicks on the link for consent.

The second set of endpoints are for the API, largely for the asynchronous OAuth flow via REST.

All endpoints begin with a root URL for the DNS Provider such as https://connect.dnsprovider.com/

They may also include any prefix at the discretion of the DNS Provider, for example, https://connect.dnsprovider.com/api/

The root URLs for the UX endpoints and the API endpoints are returned in the JSON payload during DNS Provider discovery.

4.5.1. Synchronous Flow

4.5.1.1. Query Supported Template

```
GET
{urlAPI}/v2/domainTemplates/
providers/{providerId}/services/{serviceId}
```

This URL can be used by the Service Provider to determine if the DNS Provider supports a specific template through the synchronous flow.

Returning a status of 200 without a body indicates the template is supported. Returning a status of 404 indicates the template is not supported.

4.5.1.2. Apply Template

```
GET
{urlAPI}/v2/domainTemplates/
providers/{providerId}/services/{serviceId}/apply?[properties]
```

This is the URL used to apply a template to a domain. It is called from the Service Provider to start the Domain Connect Protocol.

This URL should be called in a new browser tab or in a popup browser window. The DNS Provider would sign the user in, verify domain ownership, and ask for confirmation of application of the template.

It is also likely that the DNS Provider would warn the user of existing settings that would change and/or services that would be disrupted as part of applying this template. The fidelity of this warning is left to the DNS Provider.

Upon completion of the application of the template the DNS Provider would close this tab or window.

Parameters/properties passed to this URL include:

- domain: This parameter contains the domain name being configured.
- name/value pairs: Any variable fields consumed by this template. The name portion of this API call corresponds to the variable(s) specified in the template and the value corresponds to the value that should be used when applying the template.
- groupId: This OPTIONAL parameter specifies the group of changes from the template to apply. If no group is specified, all changes are applied. Multiple groups can be specified in comma delimited

format.

- sig: An optional signature of the query string. See Security Considerations section.

An example query string is below:

GET https://webconnect.dnsprovider.com/v2/domainTemplates/providers/coolprovider.com/services/hosting/apply?www=192.168.42.42&m=192.168.42.43&domain=example.com

This call indicates that the Service Provider wishes to connect the domain example.com to the service using the template identified by the composite key of the provider (coolprovider.com) and the service owned by them (hosting). In this example, there are two variables in this template, "www" and "m" which both require values (in this case each requires an IP address). These variables are passed as name/value pairs.

4.5.1.3. Security Considerations

By applying a template with parameters, there is a security consideration that must be taken into account

Consider an email template where the IP address of the MX record is a passed in variable. A bad actor could generate a URL with a bad IP. If an end user is convinced to click on this URL (say in a phishing email), they would land on the DNS Provider site to confirm the change. To the user, this would appear to be a valid request to configure the domain. Yet the IP would be hijacking the service.

Not all templates have this problem. But when they do, there are two options.

One option would be to not enable the synchronous flow and use asynchronous OAuth. But as will be seen below, OAuth has both a higher implementation burden and requires onboarding between each Service and DNS Provider.

As another option, digitally signing the query string will be

enabled. The signature will be appended as an additional query string parameter, properly URL encoded and of the form:

```
sig=NLOQQm6ikGC2FlFvFZqIFNCZqlaC4B%2FQDwS6iCwIElMWhXMgRnRE17zhLtdLFie
WkyqKa4I%2FOoFaAgd%2FAl%2ByzDd3sM2X1JVF5ELjTlj84jZ4KOEIdnbgkEeO%2FTkY
RrPkwcmcHMwc4RuX%2Fqio8vKYxJaKLKeVGpUNSKo7zkq3XIRgyxoLSRKxmlSTHFAz4Lv
YXPWo6SHDoVcRvElWj18Um13sSXuX4KhtOLym2yImHpboEi4m2Ziigc%2BNHZE0VvHUR7
wZgDaB01z8hFm5ATF%2B8swjandMRf2Lr4Syv4qTxMNT61r62EWFkt5t9nhxMgss6z4pf
DVFZ3vYwSJDGuRpEQ%3D%3D
```


The Service Provider can generate this signature using a private key. The DNS Provider can then verify the signature using the public key.

key=_dcpubkeyv1

This example indicates that the public key can be found by doing a DNS query for a TXT record called _dcpubkeyv1.

Since the public key may be greater than 255 characters, multiple TXT records may exist for the DNS TXT query. For a public key of:

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1dCqv7JEzUOfbhWKB9mTRsv3O
9Vzy1Tz3UQlIDGpnVrTPBJDQTXUhxUMREEOBKo+rOjHZqfYnSmlkgu1dnBEO8bsELQL8G
jS4zsjdA53gRk2SDxuzcB4fK+NCDfnRHut5nG0S3U4cq4DuGrMDFVBwxH1duTsqDNgIOO
fNTsFcWSVXoSSTqCCMGbj8Vt51umDhWQAj06lf50qP2/
jMNs2G+KTlk3dBHx3wtqYLvdcop1Tk5xBD64BPJ9uwm8KlDNHe+8O+cC9j04Ji8B2K0/
PzAj90xnb8XJy/
EM124hpT9lMgpHKBUvdeurJYweC6oP41gsTf5LrpjnyIy9j5FHPCQIDAQAB
```

There would be several TXT records. The records would be of the form:

```
p=1,a=RS256,d=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1dCqv7JE
zUOfbhWKB9mTRsv3O9Vzy1Tz3UQlIDGpnVrTPBJDQTXUhxUMREEOBKo+rOjHZqfYnS
mlkgu1dn
p=2,a=RS256,d=BEO8bsELQL8GjS4zsjdA53gRk2SDxuzcB4fK+NCDfnRHut5nG0S3
U4cq4DuGrMDFVBwxH1duTsqDNgIOOfNTsFcWSVXoSSTqCCMGbj8Vt51umDhWQAj06l
f5
p=3,a=RS256,d=NCDfnRHut5nG0S3U4cq4DuGrMDFVBwxH1duTsqDNgIOOfNTsFcWS
VXoSSTqCCMGbj8Vt51umDhWQAj06lf50qP2/
jMNs2G+KTlk3dBHx3wtqYLvdcop1Tk5xBD64BPJ9
p=4,a=RS256,d=uwm8KlDNHe+8O+cC9j04Ji8B2K0/PzAj90xnb8XJy/
```

EM124hpT9lMgpHKBUvdeurJYweC6oP41gsTf5LrpjnyIy9j5FHPCQIDAQAB

Here the public key is broken into four records in DNS, and the data also indicates that the signing algorithm is an RSA Signature with SHA-256.

It should be noted that the above data was generated for a query string:

a=1&b=2&ip=10.10.10.10&domain=foobar.com

Support for signing the query string and verification is optional. Not all services require this level of security, and not all DNS Providers will support this signing for the synchronous flow.

There are circumstances where the Service Provider may wish to verify that the template was successfully applied. Without domain connect, this typically involved the Service Provider querying DNS to see if the settings had been applied.

This same technique works with Domain Connect, and if necessary can be triggered either manually on the Service Provider site or automatically upon page/window activation in the browser.

Automatic notification via callback URLs were considered in earlier drafts, and subsequently dropped due to their lack of reliability and difficulty in getting a consistent implementation across DNS Providers.

4.5.2. Asynchronous Flow: OAuth

Using the OAuth flow is a more advanced use case, needed by Service Providers that have more complex configurations that may require multiple steps and/or are asynchronous from the user's interaction.

Details of an OAuth implementation are beyond the scope of this specification. Instead, an overview of how OAuth fits with Domain Connect is given here.

Service providers wishing to use the OAuth flow must register as an

OAuth client with the DNS Provider. This is envisioned as a manual process.

To register, the Service Provider would provide (in addition to their template) one or more callback URLs that specify where the customer will be redirected after the provider authorization. In return, the DNS Provider will give the Service Provider a client id and secret which will be used when requesting tokens as part of the OAuth process flow.

4.5.2.1. Getting an Authorization Code

```
GET
{urlAsyncUX}/v2/domainTemplates/
providers/{providerId}/services/{serviceId}
```

To initiate the OAuth flow the Service Provider would link to the DNS Provider to gain consent.

This endpoint is similar to the synchronous flow described above, and will handle authenticating the user, verification of domain ownership, and asking for the user's permission to allow the Service Provider to make the specified changes to the domain. Similarly, the

DNS Provider will often want to warn the user that (eventual) application of this template might change existing records and/or disrupt existing services attached to the domain.

While the variables for the applied template would be provided later, the values of some variables are often necessary in the consent flow to determine conflicts. As such, any variables impacting conflicting records needs to be provided in the consent flow. Today this includes variables in hosts, and variables in the data portion for certain TXT records. As conflict resolution evolves, this list may grow.

Upon successful authorization, verification, and consent, the DNS Provider will direct the end user's browser to the redirect URI provided in the request, appending the authorization code as a query parameter of "code".

Upon error, the DNS Provider will direct the end user's browser to

the redirect URI provided in the request, appending the error code as a query parameter "error".

The following describes the values to be included in the query string parameters for the request for the OAuth consent flow.

- domain: This parameter contains the domain name being configured.
- client_id: This is the client id that was provided by the DNS Provider, to the Service Provider during registration.
- redirect_uri: The location to direct the client's browser to upon successful authorization, or upon error.
- response_type: OPTIONAL. If included should be the string 'code' to indicate an authorization code is being requested.
- scope: This is the name of the template that is being requested.
- state: OPTIONAL but recommended. This is a random, unique string passed along to prevent CSRF. It will be returned as a parameter when redirecting to the redirect_url described above.
- name/value pairs: Required for fields that impact the conflict detection. This includes variables used in hosts and data in TXT records.

4.5.2.2. Requesting an Access Token

POST {urlAPI}/v2/OAuth/access_token

Once authorization has been granted the Service Provider must use the Authorization Code provided to request an Access Token. The OAuth specification recommends that the Authorization Token be a short lived token, and a reasonable recommended setting is ten minutes. As

such this exchange needs to be completed before that time has expired or the process will need to be repeated.

This token exchange is done via a server to server API call from the Service Provider to the DNS Provider.

The Access Token granted will also have a longer lifespan, but also can expire. To get a new access token, the Refresh Token is used.

The following describes the POST parameters to be included in the request.

- code: The authorization code that was provided in the previous step when the customer accepted the authorization request, or the refresh_token for a subsequent access token.
- redirect_uri: OPTIONAL. If included, needs to be the same redirect uri provided in the previous step, simple for verification.
- grant_type: The type of code in the request. Usually the string 'authorization_code' or 'refresh_token'.
- client_id: This is the client id that was provided by the DNS Provider, to the Service Provider during registration.
- client_secret: The secret provided to the Service Provider during registration.

Upon successful token exchange, the DNS Provider will return a response with 4 properties in the body of the response.

- access_token: The access token to be used when making API requests.
- token_type: Always the string "bearer".
- expires_in: The number of seconds until the access_token expires.
- refresh_token: The token that can be used to request new access tokens when this one has expired.

4.5.2.3. Making Requests with Access Tokens

Once the Service Provider has the access token, they can call the DNS Provider's API to make change to DNS on behalf of the user.

All calls to this API pass the access token in the Authorization Header of the request to the call to the API. More details can be found in the OAuth specifications, but as an example:

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```


4.5.2.4. Apply Template to Domain

```
POST
{urlAPI}/v2/domainTemplates/
```

providers/{providerId}/services/{serviceId}/apply?[properties]

The primary function of the API is to apply a template to a customer domain.

While the providerId and serviceId are also implied in the authorization, these are on the path for consistency with the synchronous flows. If not matching what is in the authorization, an error would be returned.

When applying a template to a domain, it is possible that a conflict may exist with previous settings. While it is recommended that conflicts be detected when the user grants consent, because OAuth is asynchronous it is possible that a new conflict was introduced by the user.

While it is up to the DNS Provider to determine what constitutes a conflict (see section on Conflicts below), when one is detected an error will be returned by default. This error will enumerate the conflicting records in a format described below.

Because the user isn't present at the time of this error, it is up the Service Provider to determine how to handle this error. Some providers may decide to notify the user. Others may decide to apply their template anyway using the "force" parameter. This parameter will bypass error checks for conflicts, and after the call the service will be in its desired state.

Calls to apply a template via OAuth require the following parameters:

- domain: This contains the domain name being configured. It must match the domain in the authorization token.
- name/value pairs: Any variable fields consumed by this template. The name portion of this API call corresponds to the variable(s) specified in the record and the value corresponds to the value that should be used when applying the template as per the implementation notes.
- groupId: This OPTIONAL parameter specifies the group of changes in the template to apply. If omitted, all changes are applied. This can also be a comma separated list of groupIds.
- force: This OPTIONAL parameter specifies that the template should be applied independently of any conflicts that may exist on the domain. This can be a value of 0 or 1.

An example call is below. In this example, it is contemplated that there are two variables in this template, "www" and "m" which both require values (in this case each requires an IP address). These variables are passed as name/value pairs.

POST
https://connect.dnsprovider.com/v2/domainTemplates/providers/coolprovider.com/services/hosting/
apply?www=192.168.42.42&m=192.168.42.43&force=1

The API must validate the access token for the Service Provider and that the domain belongs to the customer and is represented by the token being presented. With these checks passing, the template may be applied to the domain after verifying that doing so would not cause an error condition, either because of problems with required variables or the current state of the domain itself (for example, already having a conflicting template applied).

Results of this call can include information indicating success, or an error. Errors will be 400 status codes, with the following codes defined.

- Success (20*): A response of an http status code of 204 indicates that call was successful and the template applied. Note that any 200 level code should be considered a success.
- Unauthorized (401): A response of a 401 indicates that caller is not authorized to make this call. This can be because the token was revoked, or other access issues.
- Error (404,422): This indicates something wrong with the request itself, such as bad parameters.
- Failed (409): This indicates that the call was good, and the caller authorized, but the change could not be applied due to a conflicting template or a domain state that prevents updates. Errors due to conflicts will only be returned when force is not equal to 1.

When a 409 is returned, the body of the response will contain details of the error. This will be JSON containing the error code, a message suitable for developers, and an array of tuples containing the conflicting records type, host, and data element.

4.5.2.5. Revert Template

POST
{urlAPI}/v2/domainTemplates/
providers/{providerId}/services/{serviceId}/revert?domain={domain}

This API allows the removal of a template from a customer domain using an OAuth request.

The provider and service name in the authorizatoin must match the values in the URL. So must the domain name on the query string.

This call must validate that the template requested exists and has been applied to the domain by the Service Provider or a warning must be returned that the call would have no effect. This call must validate that there is a valid authorization token for the domain passed in or an error condition must be reported.

An example query string might look like:

POST https://connect.dnsprovider.com/v2/domainTemplates/providers/coolprovider.com/services/hosting/revert?domain=example.com

Response codes are identical to above.

4.5.2.6. Revoke Access

Like all OAuth flows, the user can revoke the access at any time using UX at the DNS Provider site. So the Service Provider needs to be aware that their access to the API may be denied.

4.6. Domain Connect Objects and Templates

Templates are not versioned. Instead, if a breaking change is made to a template it is recommended that a new template be created. While on the surface versioning looks appealing, the reality is that the settings in a template rarely change. This is because a successful service will have many customers with settings in their DNS, some applied by templates using this protocol, and some manually applied. As such changes to the template need to be done in a manner that accounts for existing customers.

A template is defined as a standard JSON data structure containing the following data:

- o providerId: The unique identifier of the Service Provider that created this template. This is used in the URLs to identify the Service Provider. To ensure non-coordinated uniqueness, this would be the domain name of the Service Provider.
- o providerName: The name of the Service Provider. This will be displayed to the user.
- o templateId: The name or identifier of the template. This is used in URLs to identify the template.

- o templateName: The friendly name of this service. This will be displayed to the user.
- o logoUrl: A graphical logo for use in any web-based flow. This is a URL to a graphical logo sufficient for retrieval.
- o description: A textual description of what this template attempts to do. This is meant to assist integrators, and therefore should not be displayed to the user.
- o syncBlock: Indicates that the synchronous protocol should not be enabled for this template.
- o synchPubKeyDomain: When present, indicates that calls to apply a template synchronously will be digitally signed. This element contains the domain name for querying a TXT record from DNS.
- o launchUrl: OPTIONAL. A URL suitable for a DNS Provider to call to initiate the execution of this template. This allows the flow to begin with the DNS Provider as described above.
- o records: A list of records for the template.

Each template record is an entry that contains a type and several optional parameters based on the value.

For all entries of a record template other than "type" and "groupId", the value can contain variables denoted by %<variable name>%. These are the values substituted at runtime when writing into DNS.

It should be noted that as a best practice, the variable should be equal to the portion of the values in the template that change as little as possible.

For example, say a Service Provider requires a CNAME of one of three values for their users: s01.example.com, s02.example.com, and s03.example.com.

The value in the template could simply contain %servercluster%, and

the fully qualified string passed in. Alternatively, the value in the template could contain s%var%.example.com. By placing more fixed data into the template, the data is more constrained.

Each record will contain the following elements:

- type: Describes the type of record in DNS, or the operation impacting DNS. Valid values include: A, AAAA, CNAME, MX, TXT, SRV, NS, APEXCNAME, REDIR301, or REDIR302.
- groupId: This OPTIONAL parameter identifies the group the record belongs to when applying changes.
- host: The host for A, AAAA, CNAME, TXT, and MX values. This is the hostname in DNS.
- pointsTo: The pointsTo location for A, AAAA, CNAME and MX records.

- ttl: This is the time-to-live for the record in DNS. Valid for A, AAAA, CNAME, TXT, MX, and SRV records.
- data: This is the data for a TXT record in DNS.
- priority: This is the priority for an MX or SRV record in DNS.
- weight: This is the weight for the SRV record.
- port: This is the port for the SRV record.
- protocol: This is the protocol for the SRV record.
- service: This is the protocol for the SRV record.

4.7. Operational and Implementation Considerations

The DNS Provider is responsible for handling of the conflicts with records already existing in the DNS Zone. This includes detection of conflicts, removing conflicts when a new template is applied, and merging records when appropriate.

For example, if the application of a template through the web based flow would interfere with previously set DNS records (either through another template or manual settings), it is envisioned that the user would be asked to confirm the clearing of the previously set template. If it would interfere with DNS records accessible through a previously issued OAuth flow, the provider could revoke the previously issued token.

Similarly, when granting an OAuth token that interferes with a previously issued OAuth token, access to the old token could

automatically be revoked.

Manual changes to DNS through the DNS Provider could have appropriate warnings in place to prevent unwanted changes; with overrides being possible and removing conflicting templates.

The behavior of these interactions is left to the sophistication of the DNS Provider. However, a general recommendation is to ensure that a newly configured service works correctly.

A proposing handling of records is as follows (if not otherwise specified, conflicts occur if the records have the same name):

Replace records of the same type for A, AAAA, MX, CNAME, APEXCNAME, SRV. If the template specifies an A or AAAA, the respective AAAA or A record should be removed to avoid IPv4 and IPv6 pointing to different services
Append to the existing records of the same type for TXT
Replace any record for CNAME
Remove any CNAME record existing at the same or parent level to any records added by the template

Additional record types and/or extensions to records in the template can be implemented on a per DNS Provider basis. However, care should be taken when defining extensions so as to not conflict with other protocols and standards. Certain record names are reserved for use in DNS for protocols like DNSSEC (DNSKEY, RRSIG) at the registry level.

Defining these optional extensions in an open manner as part of this specification is highly recommended. The following are the initial optional extensions a DNS Provider/Service Provider may support.

Some Service Providers desire the behavior of a CNAME record, but in the apex record. This would allow for an A Record at the root of the domain but dynamically determined at runtime.

The recommended record type for DNS Providers that wish to support this an APEXCNAME record. Additional fields included with this record would include pointsTo and TTL.

Defining a standard for such functionality in DNS is beyond the scope of this specification. But for DNS Providers that support this functionality, using the same record type name across DNS Providers allows template reuse.

Some Service Providers desire a redirection service associated with the A Record. A typical example is a service that requires a redirect of the domain (e.g. example.com) to the www variant (www.example.com). The www would often contain a CNAME.

Since implementation of a redirection service is typically simple, it is recommended that service providers implement redirection on their own. But for DNS Providers that have a redirection service, supporting simple templates with this functionality may be desired.

While technically not a "record" in DNS, when supporting this optional functionality, it is recommended that this be implemented using two new record types.

REDIR301 and REDIR302 would implement 301 and 302 redirects respectively. Associated with this record would be a single field called the "target", containing the target domain of the redirect.

Several service providers have asked for functionality supporting an update to the nameserver records at the registrar associated with the domain.

This functionality is again deemed as optional and up to the DNS Provider to determine if they desire to support this functionality.

When implementing this, two records will be provided. NS1 and NS2, each containing a pointsTo argument.

Requests have also been made to allow for updates to the DS record for DNSSEC. This record is required at the registry to enable DNSSEC, but can only be written by the registrar.

Note that the registrar may or may not be the DNS Provider, but in this case the implementation of updates of the DS record into the registry would be handled exclusively by the registrar.

For DNS Providers that support this record, the record type should be

DS. Values will be keyTag, algorithm, digestType, and digest.

Variables in templates that are hard-coded host names are the responsibility of the DNS Provider to protect. That is, DNS Providers are responsible for ensuring that host names do not interfere with known values (such as m. or www. or mail.) or internal names that provide critical functionality that is outside the scope of this specification.

This template format is intended for internal use by a DNS Provider and there are no codified API endpoints for creation or modification of these objects. API endpoints do not use this object directly. Instead, API endpoints reference a template by ID and then provide key/value pairs that match any variable values in these record objects.

However, by defining a standard template format it is believed it will make it easier for Service Providers to share their provisioning across DNS Providers. Further revisions of this specification may include a repository for publishing and consuming these templates. For now, templates are maintained at http://domainconnect.org.

Implementers are responsible for data integrity and should use the record type field to validate that variable input meets the criteria for each different data type.

Some considerations are necessary for configuring a domain (example.com) vs. a sub-domain (sub.example.com) for a Service.

The DNS Provider will only implement the _domainconnect record at the domain level. This means that during discovery, the Service Provider would need to call the root domain for this information.

The DNS Provider should support configuring services on domains vs. sub-domains.

If the template is identical for the root and for the sub-domain, the Service Provider simply needs to call domain connect with the fully qualified domain name. Here passing in sub.example.com vs. example.com to the domain connect flow is all that is necessary.

If there are differences, two templates would be created and the Service Provider would invoke the appropriate version.

It is also highly recommended that this approach be taken, vs. variables for host names passed into the template.

Example Records: Single static host record

Consider a template for setting a single host record. The records section of the template would have a single record of type "A" and could have a value of:

```
[{
''type'': ''A'',
''host'': ''www'',
''pointsTo'': ''192.168.1.1'',
''ttl'': 600
}]
```

This would have no variable substitution and the application of this template to a domain would simply set the host name "www" to the IP address "192.168.1.1"

Example Records: Single variable host record for A

In the case of a template for setting a single host record from a variable, the template would have a single record of type "A" and could have a value of:

```
[{
''type'': ''A'',
''host'': ''@'',
''pointsTo'': ''192.168.1.%srv%'',
''ttl'': 600
}]
```

A query string with a key/value pair of

srv=8

would cause the application of this template to a domain to set the host name for the apex A record to the IP address "192.168.1.8" with a TTL of 600.

5. IANA Considerations

5.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. The following URI assignment is requested of IANA:

URI: ietf:params:xml:ns:validate-1.0

Registrant Contact: See the "Author's Address" section of this document.

6. Acknowledgements

The authors wish to thank the following persons for their feedback and suggestions:

- Chris Ambler of GoDaddy Inc.
- Jody Kolker of GoDaddy Inc.

7. Change History

7.1. Change from 02 to 03

Added width/height JSON values returned by DNS Provider Discovery. Corrected text of GET method for getting the authorization token. Added clarifying text to Group ID description parameter of the apply template POST method. Quite a few minor edits and clarifications that were found during implementation, especially in the Implementation Considerations section.

7.2. Change from 01 to 02

Added new GET method for Service Providers to determine if the DNS Provider supports a specific template. Some other minor edits for clarification.

7.3. Change from 00 to 01

Minor edits and clarifications found during implementation.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

Authors' Addresses

Arnold Blinn
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale, AZ 85260
US

Email: arnoldb@godaddy.com
URI: http://www.godaddy.com

Roger Carney
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale, AZ 85260
US

Email: rcarney@godaddy.com
URI: http://www.godaddy.com