Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

byndns

CLI e REST API per l'aggiornamento dinamico dei record DNS

Angelo Esposito[1], Paolo Vanacore[2]

[1]Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche (ICAR-CNR)

Via Pietro Castellino, 111 – 80131 Napoli

[2]Area Territoriale di Ricerca di Napoli 1

Via Pietro Castellino, 111 – 80131 Napoli

angelo.esposito@cnr.it

paolo.vanacore@cnr.it

Versione 1.0

RT-ICAR-NA-2026-05 **Febbraio 2026**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: www.icar.cnr.it

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

byndns

CLI e REST API per l'aggiornamento dinamico dei record DNS

[1]Angelo Esposito, [2]Paolo Vanacore

[1]Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche (ICAR-CNR) Via Pietro Castellino, 111 – 80131 Napoli

[2] Area Territoriale di Ricerca di Napoli 1

Via Pietro Castellino, 111 – 80131 Napoli

angelo.esposito@icar.cnr.it, paolo.vanacore@cnr.it

Abstract

Il presente rapporto tecnico descrive la progettazione e l'implementazione di byndns, una soluzione software sviluppata per servizi di Dynamic DNS (DDNS) su infrastrutture basate sul server DNS BIND. In contesti moderni caratterizzati da mobilità e cloud computing, dove gli indirizzi IP dei client variano frequentemente, la gestione statica dei record DNS risulta talvolta inadeguata. Byndns risponde a questa esigenza, offrendo un'interfaccia CLI per l'integrazione con script batch e una API REST per l'integrazione con sistemi esterni. Il sistema rappresenta un wrapper, conforme allo standard RFC 2136. Le comunicazioni sono crittografate su HTTPS ed è prevista l'autenticazione tramite Bearer Token per i client delle API. Inoltre, attraverso firme TSIG è garantita l'integrità delle transazioni verso il server BIND. Il documento illustra

l'architettura del sistema, le tecnologie impiegate (Python, BIND) e i processi di deploy, configurazione e utilizzo.

Keywords: DNS, DDNS, RFC 2136, REST API

1. Introduzione

L'obiettivo di questo documento è descrivere l'architettura, le funzionalità e le modalità di utilizzo di byndns, un software progettato per implementare un servizio di Dynamic DNS [1], integrato con il server DNS BIND [2]. Il documento fornisce inoltre le istruzioni per l'installazione, la configurazione e il deploy del sistema, nonché esempi pratici di utilizzo tramite interfaccia a riga di comando (CLI) e API REST.

La gestione dinamica dei record DNS è un requisito fondamentale in scenari in cui gli indirizzi IP dei client cambiano frequentemente, come nel caso di infrastrutture cloud e dispositivi mobile.

byndns nasce dall'esigenza di fornire un'interfaccia semplice e sicura per effettuare aggiornamenti DNS dinamici, sfruttando le potenzialità di BIND e attraverso meccanismi di autenticazione e comunicazione sicura.

byndns consente di aggiornare dinamicamente i record DNS attraverso due modalità principali:

- **CLI (Command Line Interface)**: per script locali;
- **REST API**: per l'integrazione con applicazioni e servizi esterni.

Le principali caratteristiche includono:

- **Compatibilità con RFC 2136**: utilizzo di nsupdate [3] e implementazione nativa in Python;
- **Flessibilità di deploy**: può essere installato sullo stesso server di BIND o su un server separato;
- **Comunicazione sicura**: utilizzo del protocollo HTTPS per le API REST;
- **Autenticazione e autorizzazione**: accesso protetto tramite token da includere nelle intestazioni delle richieste delle API REST.

2. Tecnologie

Il Domain Name System (DNS) è il sistema che traduce i nomi di dominio in indirizzi IP, consentendo la comunicazione tra dispositivi su Internet. In scenari statici, i record DNS vengono configurati manualmente e modificati di rado. Tuttavia, in contesti dinamici (es. dispositivi mobili, ambienti cloud, connessioni domestiche), gli indirizzi IP possono cambiare frequentemente, rendendo necessaria una soluzione per aggiornare automaticamente i record DNS. Il Dynamic DNS (DDNS) risolve questo problema, permettendo l'aggiornamento in tempo reale dei record DNS, quando l'indirizzo IP associato a un host cambia.

Lo standard RFC 2136 definisce il protocollo per effettuare aggiornamenti dinamici ai record DNS. Questo approccio consente di aggiungere, modificare o eliminare record. Gli aggiornamenti RFC 2136 sono supportati da BIND e possono essere effettuati tramite strumenti come nsupdate, che utilizza il protocollo DNS over UDP/TCP per inviare le richieste al server autoritativo.

BIND (Berkeley Internet Name Domain) è uno dei server DNS più diffusi e supporta nativamente gli aggiornamenti dinamici RFC 2136. Per abilitare questa funzionalità, è necessario configurare le zone DNS come dinamiche, definire le policy di sicurezza tramite TSIG keys (Transaction Signatures) [4] per autenticare le richieste di aggiornamento e abilitare i permessi di aggiornamento nella configurazione del server. BIND è altamente configurabile e rappresenta la base ideale per implementare un servizio di Dynamic DNS sicuro e scalabile.

3. Architettura di byndns

Le componenti principali di byndns sono:

- *REST API Server* che espone le REST API per creare/aggiornare/eliminare record DNS;
- *modulo CLI* per l'integrazione di script locali;
- *core* di byndns che costruisce le operazioni di update attraverso un adapter per le comunicazioni con BIND, secondo l'RFC 2136. Gli adapter utilizzabili, scelti in fase di configurazione, sono:
    - nsupdate – byndns genera i comandi per nsupdate e lo invoca creando un sotto-processo;
    - python – utilizza un'implementazione python dell'RFC 2136 (libreria dnspython [5]).

In Figura 1 è riportato uno schema dell'architettura del software realizzato.

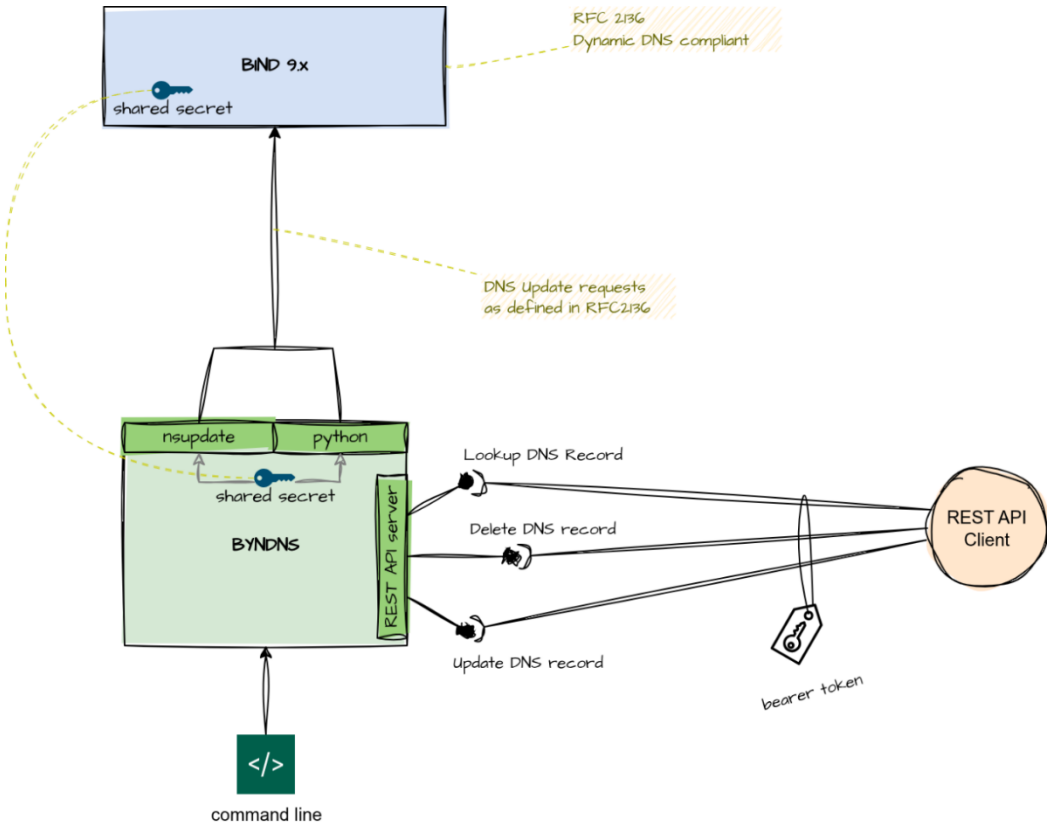

Figura 1 architettura di byndns

Indipendentemente dall'adapter usato (dnsupdate/python), per consentire la verifica delle origini e dell'integrità degli aggiornamenti delle zone DNS, la comunicazione tra byndns e BIND è autenticata attraverso chiave condivisa TSIG (Transaction Signature).

Le comunicazioni tra client REST e API Server, invece, fanno uso del protocollo di crittografia TLS su protocollo di comunicazione HTTP (HTTPS), mentre, l'autenticazione/autorizzazione dei client avviene attraverso API Token.

4. Deploy

Il software è stato testato con la seguente configurazione:

- BIND >= 9.18
- Python 3.10
- Pip 23.0.1

Nei successivi sotto paragrafi viene illustrato il processo di deploy che consiste nel configurare BIND, al fine di abilitare la gestione delle richieste remoto di aggiornamento dinamico, nell'installare e configurare byndns.

4.1 Configurazione di BIND

Per garantire l'integrazione di byndns con BIND è necessario configurare quest'ultimo affinché accetti richieste remote di aggiornamenti dinamici [6].

A scopo di esempio, si suppone che la nuova zona dinamica sia `dyn.myzone.com`, si riportano di seguito i passi da seguire:

Creazione di una cartella per la zona dinamica e impostazione del proprietario

```
mkdir /etc/bind/dynamic
chown bind:bind /etc/bind/dynamic
```

1) Creazione del file per la nuova zona dinamica

```
vim /etc/bind/dynamic/dyn.myzone.com
```

Il file dovrà avere il seguente contenuto

```
$ORIGIN      .
$TTL 30      ; 30 seconds
dyn.myzone.com              IN SOA  ns.myzone.com. hostmaster.myzone.com. (
                            2016062801 ; serial
                            3600       ; refresh (1 hour)
                            600        ; retry (10 minutes)
                            2600       ; expire (43 minutes 20 seconds)
                            30         ; minimum (30 seconds)
                            )
                    NS      ns.myzone.com.
                    NS      ns2.myzone.com.
```

2) Impostazione dell'utente e dei gruppi proprietari del file di zona

```
chown bind:bind /etc/bind/dynamic/dyn.myzone.com
```

3) Generazione dello shared secret per il record TSIG usato per verificare l'origine delle richieste di aggiornamento dinamico

```
tsig-keygen -a hmac-sha256 dyn.myzone.com >> /etc/bind/dns.keys.conf
```

4) Configurazione della zona dinamica – aggiunta del blocco `/etc/bind/named.conf`

```
include "/etc/bind/dns.keys.conf";
zone "dyn.myzone.com" {
     type master;
     file "/etc/bind/dynamic/dyn.myzone.com";
     allow-update { key "dyn.myzone.com"; };
};
```

5) Reload del servizio per il re-scan delle nuove zone

```
rndc reload
```

E' possibile testare la nuova configurazione di BIND attraverso il comando `nsupdate`:

- Aggiunta di nuovo record

```
nsupdate -k /etc/bind/dns.keys.conf
> server localhost
> zone dyn.myzone.com
> update delete test01.dyn.myzone.com A
> update add test01.dyn.myzone.com 86400 A 192.168.1.1
> send
> quit
```

- Interrogazione del DNS per verificare il nuovo record inserito

```
nslookup dyn.myzone.com localhost
```

4.2 Installazione e configurazione di byndns

Il pacchetto software include il file dei requisiti Python, `requirements.txt`. È possibile installare le dipendenze, eventualmente all'interno di un ambiente virtuale Python, utilizzando PIP:

```
pip install -r requirements.txt
```

Nella directory root del progetto è presente il file `config.yaml` contenente i parametri di configurazione di byndns:

```
server:                          ## Server configuration ##
  port: 8443                     # Port for the API server
  ssl_certfile: "certs/cert.pem" # Path to the SSL certificate file
  ssl_keyfile: "certs/key.pem"   # Path to the SSL key file
  base_path: "/ddns/api/v1"      # base path for the API

auth:                             ## Authentication configuration ##
```

```yaml
  valid_tokens:                    # List of valid tokens for authentication
    - token: "supersecrettoken123" # This is a super secret token: can update any record
    - token: "anothertoken456"     # This is another token: can only update specific records
      updatable_domains:
        - "host2.dyn.example.com"
        - "testapi.dyn.isasi.cnr.it"

dns:                               ## DNS configuration ##
  server: "192.168.1.70"           # DNS server address
  port: 53                         # DNS server port
  zone: "dyn.example.com"          # DNS zone
  keyfile: "dns.keys.conf"         # Path to the DNS key file
  RFC2136-mode: "python"           # Mode for DNS updates, can be 'nsupdate' or 'python'
```

Di seguito si riporta una descrizione di tali parametri:

- **server** – sezione dei parametri per il server REST API
  - port: numero di porta su cui l'API è in ascolto per le richieste REST;
  - ssl_certfile: path al certificato SSL per l' HTTPS;
  - ssl_keyfile: path al file contenente la chiave privata SSL;
  - base_path: basepath per gli URL degli endpoint;
- **auth** – sezione per l'autenticazione delle richieste REST API
  - valid_tokens: lista di token accettati per l'autenticazione delle richieste REST;
    - token:
    - updatable_domains [opzionale per ogni token]: lista di domini che possono essere aggiornati utilizzando il token;
- **dns** – sezione per le configurazioni relative al server DNS
  - server: indirizzo/hostname del server DNS a cui inviare le richieste (BIND);
  - port: porta su cui in ascolto il server DNS;
  - zone: zona del DNS a cui appartengono i domini da aggiornare;
  - keyfile: path al file chiave DNSSEC per gli aggiornamenti sicuri – NOTA: Il file fornisce il segreto condiviso con BIND, necessario per l'autenticazione delle richieste di aggiornamento DNS dinamico. Questo file deve essere generato sul server BIND e memorizzato sia sull'host su cui è in esecuzione byndns che sul server BIND. Per i dettagli sulla generazione di questo file, fare riferimento al paragrafo 4.1 "Configurazione di BIND".
  - RFC2136-mode: metodo utilizzato per l'invio delle richieste di aggiornamento DNS. I possibili valori sono: `nsupdate`; `python`. L'utilizzo di `nsupdate` richiede la disponibilità, sul sistema operativo in cui è in esecuzione byndns, del comando nsupdate e il relativo puntamento nella variabile PATH di sistema.

5. Esempi d'uso

5.1 CLI

L'utility CLI di byndns, `cli.py`, consente di gestire i record DNS direttamente da riga di comando. Lo strumento fornisce comandi per aggiornare, eliminare e interrogare i record all'interno di una specifica zona DNS (si faccia riferimento al file config.yaml per la configurazione della zona, nonché a quanto documentato nel Paragrafo 4.2).

Il comando per eseguire lo script Python ha la seguente forma:

```
python cli.py <command> [options]
```

I comandi supportati sono:

update: inserimento o aggiornamento di un record;

delete: cancellazione di un record;

lookup: Lookup di un record.

Le opzioni possono essere:

--name: nome del record (obbligatorio)

--type: tipo di record (A, AAAA, ...) (obbligatorio)

--value: valore del record (richiesto per il comando update)

--ttl: valore del Time to live in secondi per il record (opzionale per il comando update – valore di default 3600)

I possibili exit codes del comando sono:

0: comando eseguito con successo ;

2: operazione fallita (update/delete);

3: errore;

4: record non trovato (lookup).

Di seguito sono riportate le strutture dei diversi comandi:

- inserimento o aggiornamento di un record

```
python cli.py update --name <record_name> --type <record_type> --value <record_value> [--ttl <ttl_seconds>]
```

- cancellazione di un record:

```
python cli.py delete --name <record_name> --type <record_type>
```

- lookup

```
python cli.py lookup --name <record_name> --type <record_type>
```

5.2 REST API

Installato e configurato byndns, come riportato nel paragrafo 4.2, è possibile avviare il server per le API Restfull con il comando:

```
python -m app.main
```

Di seguito si riportano gli schemi dei dati JSON per la validazione dei documenti scambiati con gli endpoint:

- **DNSMeRecord** – oggetto che modella le informazioni necessarie all'aggiornamento di un record DNS relativo all'IP del chiamante:

```
DNSMeRecord object
    name   string
    ttl    integer
    type   string
```

- **DNSRecord** – oggetto che modella le informazioni necessarie all'aggiornamento di un record DNS:

```
DNSRecord object
    name   string
    ttl    integer
    type   string
    value  string
```

- **HTTPValidationError** – oggetto che rappresenta un errore HTTP 422 (Unprocessable Entity) contenente uno o più dettagli di errori di validazione:

```
HTTPValidationError object
    detail array<object>
        Items object
            loc* array<(string | integer)>
                Items (string | integer)
                    Any of (string | integer)
                        #0 string
                        #1 integer
            msg*  string
            type* string
```

- **ValidationError** – oggetto che rappresenta un errore di validazione:

```
ValidationError object
    loc* array<(string | integer)>
        Items (string | integer)
            Any of (string | integer)
                #0 string
                #1 integer
    msg*  string
```

```
type*  string
```

Di seguito, sono riportati gli endpoint per l'utilizzo del servizio REST e relativi esempi usando il comando cURL [7]:

- **DNS lookup**

Richiesta	
Method	GET
Path	/
Descr.	Cerca un record DNS esistente
Tipo di autenticazione	Bearer Token
Payload (JSON body)	DNSRecord `object`

Risposta		
HTTP status code	**Descrizione**	**Payload**
200	Successo	DNSRecord `object`
403	Permesso negato	-
422	Errore di Validazione	ValidationError `object`

Esempio: lookup del record A per il nome a dominio `testapi.dyn.icar.cnr.it`

```
curl --location --request GET 'https://localhost:8443/ddns/api/v1' \
--header 'Content-Type: application/json' \
--header 'Authorization: ••••••' \
--data '{
    "name": "testapi.dyn.icar.cnr.it",
    "type": "A"
}'
```

- **Insert/Update**

Richiesta	
Method	POST
Path	/
Descr.	Inserisce/Aggiorna un record DNS
Tipo di autenticazione	Bearer Token
Payload (JSON body)	DNSRecord `object`

Risposta		
HTTP status code	**Descrizione**	**Payload**
200	Successo	-
403	Permesso negato	-
422	Errore di Validazione	ValidationError `object`

Esempio: inserimento del record A per il nome a dominio `testapi.dyn.icar.cnr.it`

```
curl --location 'https://localhost:8443/ddns/api/v1' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer ••••••' \
```

```
--data '{
    "name": "testapi.dyn.icar.cnr.it",
    "type": "A",
    "ttl": 4600,
    "value": "192.168.1.102"
}'
```

- **Delete**

Richiesta	
Method	DELETE
Path	/
Descr.	Elimina un record DNS
Tipo di autenticazione	Bearer Token
Payload (JSON body)	DNSRecord `object`

Risposta		
HTTP status code	**Descrizione**	**Payload**
200	Successo	-
403	Permesso negato	-
422	Errore di Validazione	ValidationError `object`

Esempio: cancellazione del record A per il nome a dominio `testapi.dyn.icar.cnr.it`

```
curl --location --request DELETE 'https://localhost:8443/ddns/api/v1' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer ••••••' \
--data '{
    "name": "testapi.dyn.icar.cnr.it",
    "type": "A"
}'
```

- **Insert or update a DNS record with the REST client IP**

Richiesta	
Method	POST
Path	/me
Descr.	Inserisce/Aggiorna un record DNS usando l'IP del client che effettua la richiesta
Tipo di autenticazione	Bearer Token
Payload (JSON body)	DNSMeRecord `object`

Risposta		
HTTP status code	**Descrizione**	**Payload**
200	Successo	-
403	Permesso negato	-
422	Errore di Validazione	ValidationError `object`

Esempio: inserimento del record A per il nome a dominio `testapi.dyn.icar.cnr.it` usando l'IP del chiamante

```
curl --location 'https://localhost:8443/ddns/api/v1/me' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer ••••••' \
--data '{
    "name": "testapi.dyn.icar.cnr.it",
    "type": "A",
    "ttl": 4600
}'
```

6. Documentazione di riferimento

Al fine di garantire la manutenibilità del sistema, il software è stato corredato da documentazione tecnica, strutturata su diversi livelli di dettaglio. Le guide introduttive, la documentazione architetturale ed esempi d'uso, sono state redatte in formato Markdown [8] (Figura 2), assicurando integrazione con il sistema di versionamento GIT adottato. Per le interfacce esposte, invece, le specifiche API sono consultabili sia tramite Swagger UI [9] (Figura 3), per l'interazione diretta e il testing degli endpoint, sia attraverso Redoc [10] (Figura 4), per una visualizzazione chiara e gerarchica delle risorse. Dopo aver avviato il server RESTfull (paragrafo 5.2), tramite browser, è possibile accedere all'interfaccia web di Swagger al link `https://< REST_SERVER_ADDR >/docs`; mentre, l'interfaccia web di Redoc è raggiungibile all'indirizzo `https://<REST_SERVER_ADDR>/redoc`.

È stata inoltre realizzata una collection Postman [11] contenente esempi d'uso delle API.

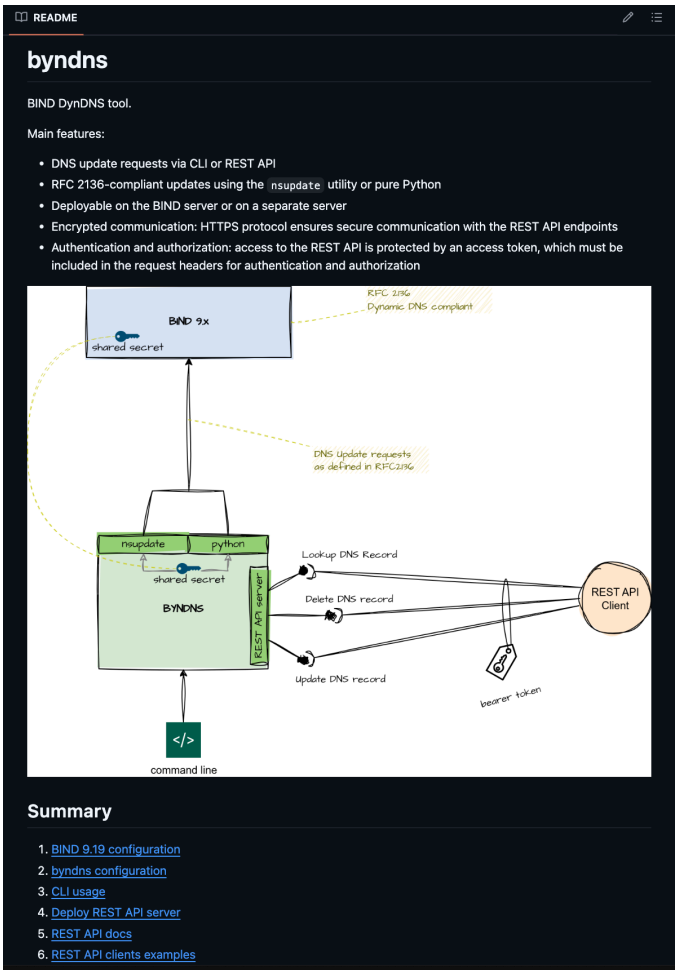

Figura 2 pagina principale (README.md) in Markdown

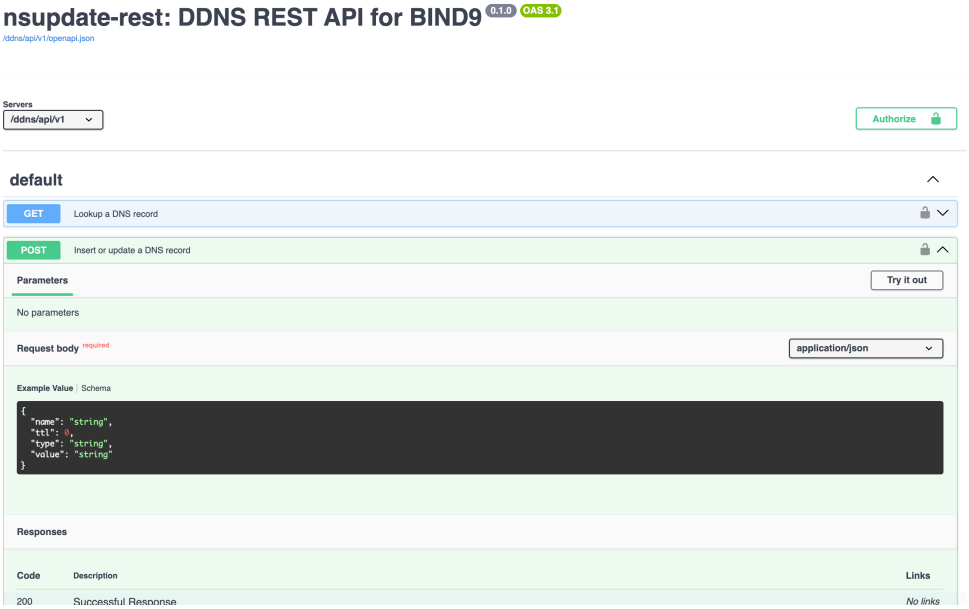

Figura 3 interfaccia web di Swagger

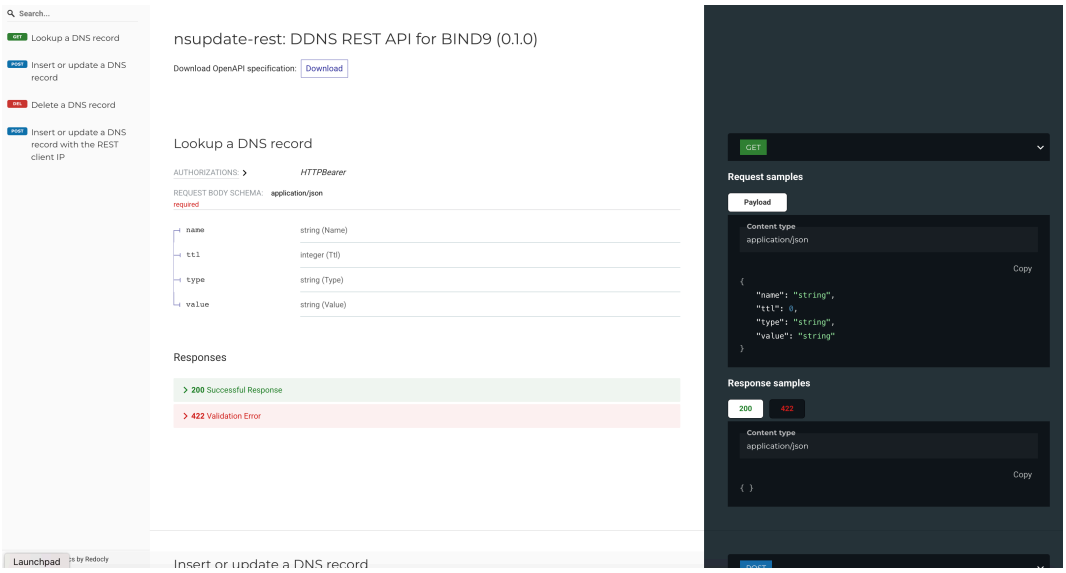

Figura 4 interfaccia web di Redoc

7. Conclusioni

L'adozione di byndns consente di aggiungere funzionalità di aggiornamento dinamico ad una infrastruttura DNS esistente, basata su BIND, arricchendo la gestione tradizionale dei name server con una soluzione che mira a soddisfare le esigenze di flessibilità richieste dagli attuali scenari di rete. L'uso di standard di sicurezza, quali HTTPS per il trasporto, i Bearer token per l'accesso e TSIG per l'interazione con BIND, assicurano l'integrità della zona. Inoltre, la documentazione prodotta, strutturata (Markdown) e interattiva (Swagger/Redoc/Postman), garantisce manutenibilità ed evoluzione del software.

8. Bibliografia

[1] RFC 2136, Dynamic Updates in the Domain Name System (DNS UPDATE) - https://www.rfc-editor.org/rfc/rfc2136
[2] BIND, Internet Systems Consortium (ISC) – https://www.isc.org/bind/
[3] nsupdate, Linux Administration and Privileged Commands Manual – https://linux.die.net/man/8/nsupdate
[4] RFC 2845, Secret Key Transaction Authentication for DNS (TSIG) – https://www.rfc-editor.org/rfc/rfc2845
[5] dnspython - https://www.dnspython.org/
[6] Configuring BIND as an RFC 2136 Dynamic DNS Server, Netgate – https://docs.netgate.com/pfsense/en/latest/recipes/bind-rfc2136.html
[7] curl, GitHub official repository – https://github.com/curl/curl
[8] Markdown website – https://daringfireball.net/projects/markdown/
[9] Swagger UI website – https://swagger.io/tools/swagger-ui/
[10] Redoc website – https://redocly.com/
[11] Postman website – https://www.postman.com/