DNSroboCert

Release 3.20.1

May 24, 2022

Table of Contents

CHAPTER 1

Introduction

Table of Contents

1.1 Features

DNSroboCert is designed to manage Let's Encrypt SSL certificates based on DNS challenges.

- Let's Encrypt wildcard and regular certificates generation by Certbot using DNS challenges,
- Integrated automated renewal of almost expired certificates,
- Standardized API through Lexicon library to insert the DNS challenge with various DNS providers,
- Centralized YAML configuration file to maintain several certificates and several DNS providers with configuration validity control,
- Modification of container configuration without restart,
- Flexible hooks upon certificate creation/renewal including containers restart, commands in containers or custom hooks,
- Support for DNS alias mode (see the `follow_cnames` option in the certificate section),
- Linux, Mac OS X and Windows support, with a particular care for Docker services,
- Delivered as a standalone application and a Docker image.

1.2 Why use DNSroboCert

If you are reading these lines, you certainly want to secure all your services using Let's Encrypt SSL certificates, which are free and accepted everywhere.

If you want to secure Web services through HTTPS, there is already plenty of great tools. In the Docker world, one can check Traefik, or nginx-proxy + letsencrypt-nginx-proxy-companion. Basically, theses tools will allow automated and dynamic generation/renewal of SSL certificates, based on TLS or HTTP challenges, on top of a reverse proxy to encrypt everything through HTTPS.

So far so good, but you may fall in one of the following categories:

1. You are in a firewalled network, and your HTTP/80 and HTTPS/443 ports are not opened to the outside world.
2. You want to secure non-Web services (like LDAP, IMAP, POP, etc.) were the HTTPS protocol is of no use.
3. You want to generate a wildcard certificate, valid for any sub-domain of a given domain.

For the first case, ACME servers need to be able to access your website through HTTP (for HTTP challenges) or HTTPS (for TLS challenges) in order to validate the certificate. With a firewall these two challenges - which are widely used in HTTP proxy approaches - will not be usable: you need to ask a DNS challenge. Please note that traefik embed DNS challenges, but only for few DNS providers.

For the second case, there is no website to use TLS or HTTP challenges, and you should ask a DNS challenge. Of course you could create a "fake" website to validate the domain using a HTTP challenge, and reuse the certificate on the "real" service. But it is a workaround, and you have to implement a logic to propagate the certificate, including during its renewal. Indeed, most of the non-Web services will need to be restarted each time the certificate is renewed.

For the last case, the use of a DNS challenge is mandatory. Then the problems concerning certificates propagation that have been discussed in the second case will also occur.

The solution is a dedicated and specialized tool which handles the creation/renewal of Let's Encrypt certificates, and ensure their propagation in the relevant services. It is the purpose of this project.

CHAPTER 2

User guide

2.1 Preparation

In order to work properly, DNSroboCert requires a configuration file, which is written in YAML. To avoid any problem, you should create it before starting DNSroboCert (the file can be empty at this point).

The path of this file is configurable. We will assume in this user guide that it is `/etc/dnsrobocert/config.yml`.

On Linux for instance, run:

```
mkdir /etc/dnsrobocert
touch /etc/dnsrobocert/config.yml
```

DNSroboCert will store the certificates in a specific folder. Here also a good idea is to create it before starting DNSroboCert.

The path of this folder is configurable. We will assume in this user guide that it is `/etc/letsencrypt`.

On Linux for instance, run:

```
mkdir -p /etc/letsencrypt
```

2.2 Installation

DNSroboCert can be installed in two ways:

1) On the host using a python package manager
2) As a Docker container

2.2.1 Installation on the host

Note:

- DNSroboCert requires Python 3.6+.
- It can be installed on Linux, Mac OS or Windows.
- For Linux and Mac OS, running as a privileged user (eg. root) is recommended.
- For Windows, running as a privileged user (eg. Administrator) is required.

The recommended way is to use Pipx, a tool that extends Pip and is specifically designed to install a Python program in a isolated and dedicated environment.

You need Python 3.6+ installed on your machine.

Install *pipx* using *pip*:

```
python3 -m pip install pipx
python3 -m pipx ensurepath
```

Then install DNSroboCert on the desired version (eg. `3.0.0`):

```
python3 -m pipx install dnsrobocert==3.0.0
```

At this point DNSroboCert is installed and available in the `PATH`. You can display the inline help using:

```
dnsrobocert --help
```

And run DNSroboCert with:

```
dnsrobocert -c /etc/dnsrobocert/config.yml -d /etc/letsencrypt
```

DNSroboCert will continue to run in the foreground. To stop it, press CTRL+C.

2.2.2 Running as a Docker container

An up-to-date Docker image is available in DockerHub. In order to persist DNSroboCert configuration and the generated certificates, you should mount its configuration and the dedicated folder for certificates from the host into the container.

- For the configuration file, expected path is `/etc/dnsrobocert/config.yml`
- For the certificates folder, expected path is `/etc/letsencrypt`

Note: Both paths are configurable in the container through the environment variables `CONFIG_PATH` and `CERTS_PATH` respectively.

Finally you can run this typical command for the desired version (eg. 3.0.0):

```
docker run --rm --name dnsrobocert
    --volume /etc/dnsrobocert/config.yml:/etc/dnsrobocert/config.yml
    --volume /etc/letsencrypt:/etc/letsencrypt
    adferrand/dnsrobocert:3.0.0
```

The Docker container will continue to run in the foreground. To stop it, press CTRL+C.

Note: DNSroboCert also work on Podman

2.3 Configuration

This guide focuses only on the bare minimum to make use of DNSroboCert: create one or more certificates. For an advanced configuration, in order to use more of DNSroboCert capabilities, please have a look to the Configuration reference.

Configuring DNSroboCert consists in writing its unique configuration file (we assume its location at `/etc/dnsrobocert/config.yml`). In particular 3 things need to be set up, and correspond to the 3 main sections of the configuration file:

- in `acme`, we define the Let's Encrypt account that will be used to issue certificates
- in `profiles`, we describe the DNS credentials and the DNS provider associated to the DNS zone to fulfill DNS-01 challenges
- and finally in `certificates`, we list the certificates that DNSroboCert will issue and regularly renew.

We can write the configuration file in draft mode: in this case, DNSroboCert will validate the configuration file, but will not do anything with it. This is quite suitable during the initial configuration phase.

So let's start with a `config.yml` whose content is:

```yaml
draft: true
```

2.3.1 Configuring `acme` section

Basically we need to decide which email will be associated to the Let's Encrypt account. This email is used by Let's Encrypt administrators to broadcast important messages, and particularly when your certificates are about to expire. This email is put in the *acme.email_account* property.

Note: During DNSroboCert configuration, you will certainly want to test things without targeting the Let's Encrypt production servers, since these servers have certificate rate creation limits. This can be done by setting the property `acme.staging` to `true`: in this case Let's Encrypt staging servers will be used.

At this point, our `config.yml` looks like this:

```yaml
draft: true
acme:
  email_account: john.doe@example.net
  staging: true
```

2.3.2 Configuring `profiles` section

It is time to set the credentials and other specific configuration entries for the DNS provider that is holding the DNS zone for the domains you want to include in your certificate. This constitute a so-called "profile" in DNSroboCert.

Please have a look to the Lexicon Providers configuration reference page to see what are the DNS providers supported by DNSroboCert (through the Lexicon tool), and what are the relevant configuration parameters for your provider.

We need to create a profile, and add it in the list holded by the `profiles` property. This profile needs:

- a name, on the property `profiles[].name`
- the Lexicon provider, as defined in the Lexicon Providers configuration reference page, on the property `profiles[].provider`
- the provider options described in the aforementioned page for your provider, exposed as an object in the `profiles[].provider_options` where each key is an option, and the value is the value option.

Typically a profile looks like the following:

```yaml
profiles:
- name: my_profile
  provider: a_provider
  provider_options:
    one_option: one_value
    another_option: another_value
```

We assume here that the `henet` provider will be used. It requires two options: `auth_username` and `auth_password`.

Given the format for `profiles`, our existing `config.yml` and the use of `henet` provider, our configuration file will look like this now:

```yaml
draft: true
acme:
  email_account: john.doe@example.net
  staging: true
profiles:
- name: henet_profile
  provider: henet
  provider_options:
    auth_username: USER
    auth_password: PASSWD
```

Note: You can declare multiple profiles to use different providers and/or the same provider with different credentials.

2.3.3 Configuring `certificates` section

Everything is ready to get the certificates. What you want as certificates is defined in the `certificates` section. It contains a list of each certificate you want. The bare minimum content for a certificate is:

- the profile name to use for DNS-01 challenges, set in the `certificates[].profile` property
- the list of domains to add in the certificate, to give as a list in the `certificates[].domains` property

Typically a certificate entry will looks like:

```yaml
certificates:
- domains:
  - one.example.net
  - two.example.net
  profile: my_profile
```

We assume here that the DNS zone is `example.net`, and two certificates need to be created:

- a regular certificate for `mail.example.net` and `ldap.example.net`
- a wildcard certificate for `*.example.net` and `example.net`

We will use the `henet_profile` configured previously.

Given this situation, we add the certificate configurations to our `config.yml`. The configuration file looks like this now:

```yaml
draft: true
acme:
  email_account: john.doe@example.net
  staging: true
profiles:
- name: henet_profile
  provider: henet
  provider_options:
    auth_username: USER
    auth_password: PASSWD
certificates:
- domains:
  - mail.example.net
  - ldap.example.net
  profile: henet_profile
- domains:
  - "*.example.net"
  - example.net
  profile: henet_profile
```

2.4 Running DNSroboCert

Our configuration is now ready: we can disable the draft mode, by setting `draft` parameter to `false`. We continue to assume that the certificates will be generated in the `/etc/letsencrypt` folder.

If DNSroboCert is already started, it will immediately proceed to issue and retrieve the certificates. If not, see the *Installation* section to start DNSroboCert.

After a minute, your certificates will be issued (have a look to the log output to check that). Your certificates are available in the `/etc/letsencrypt` folder and can be used. The layout of `/etc/letsencrypt` follows the Certbot layout convention. So given our example here, you will find:

- the regular certificate for `mail.example.net` and `ldap.example.net` at `/etc/letsencrypt/live/mail.example.net`
- the wildcard certificate for `*.example.net` and `example.net` at `/etc/letsencrypt/live/example.net`

Note: If you used the Let's Encrypt staging servers to configure DNSroboCert, you can now go back to th production servers to get real certificates: in `config.yaml`, change `acme.staging` value to `false`. DNSroboCert will proceed immediately to replace the testing certificates by real certificates.

2.4.1 Dynamic configuration

DNSroboCert check constantly for modifications in its configuration file. You can live edit it: DNSroboCert will proceed to issue new certificates as soon as you configuration file is written to the disk.

2.4.2 Automated renewal

Let's Encrypt certificates last only 3 months, and need to be renewed regularly. DNSroboCert includes this functionality: while it is running it will regularly (twice a day) check for certificate renewal, and proceed to all renewals if needed (this happens typically one month before the expiration of the current certificate).

2.4.3 Daemonize DNSroboCert

Because of this regular renewal requirement, DNSroboCert should run constantly on your machine as a daemon. The tool does not provide a specific daemon technology: the CLI will just constantly run on the foreground, and reacts properly to the relevant exit signal codes like `SIGTERM`. From that it is your reponsability to daemonize DNSroboCert.

Here are some relevant ways depending on the context.

Systemd unit

If you run DNSroboCert directly on the host (eg. you followed the *Installation on the host* section), one simple way is to define a systemd unit, and configure your Systemd to run DNSroboCert as a daemon at startup.

Docker-Compose

If you run DNSroboCert in a Docker container (eg. you followed the *Running as a Docker container* section), then Docker-Compose is a standard way to configure a Docker and ensure that is runs all the time as a daemon.

Create the following `docker-compose.yml` file:

```yaml
version: '2'
services:
  dnsrobocert:
    image: adferrand/dnsrobocert
    container_name: dnsrobocert
    volumes:
    - /etc/letsencrypt:/etc/letsencrypt
    - /etc/dnsrobocert:/etc/dnsrobocert
    restart: always
```

Then run it:

```
docker-compose up -d
```

At this point, your Docker container of DNSroboCert will be started and the Docker daemon will ensure it continues to run upon your machine restart.

2.4.4 Run DNSroboCert in "one-shot" mode

If the approach of DNSroboCert process running constantly does not fit your needs, you can also use the "one-shot" mode.

In this mode, DNSroboCert will process only once the provided configuration upon execution, then:

- create or update certificates if needed
- renew expired certificates
- delete certificates that do not match the current configuration

At the end it will exit immediately without setting up any config watch or automated renewal process: it will be up to you to execute DNSroboCert on a regular basis (preferably twice a day as recommended by Let's Encrypt).

To use the "one-shot" mode, simply set the *–one-shot* flag to the command line. For instance:

```
dnsrobocert --config /path/to/config.yml --directory /path/to/letsencrypt --one-shot
```

CHAPTER 3

Configuration reference

DNSroboCert configuration is defined in a central file, usually located at `/etc/dnsrobocert/config.yml`. Its location is defined by the `-c` flag when running DNSroboCert locally with the CLI, or with the `CONFIG_PATH` environment variable with Docker.

3.1 File format

The configuration file must be a valid YAML and must conform to the JSON schema defined for DNSroboCert. One can find the raw content of this schema on GitHub. DNSroboCert will validate the file each time it is changed, and output the errors (missing properties, wrong value type) if any.

The basic structure is the following:

```yaml
draft: false
acme: {}
profiles: []
certificates: []
```

3.2 `draft` Section

If the draft mode is enabled, DNSroboCert will validate dynamically the configuration file, but will not reconfigure itself with it and will not proceed to any further action. This is useful to make wide modifications in the file without DNSroboCert taking them into account immediately, then apply all modifications altogether by disabling the draft mode.

```yaml
draft: true
```

3.3 acme Section

This section contains all general configuration parameters for Certbot (the underlying ACME client that generates the certificates) and how these certificates are stored locally.

```
acme:
  email_account: my.email@example.net
  staging: false
  api_version: 2
  # If directory_url is set, values of staging and api_version are ignored
  directory_url: https://example.net/dir
  certs_permissions:
    files_mode: 0644
    dirs_mode: 0755
    user: nobody
    group: nogroup
  crontab_renew: 12 01,13 * * *
```

3.3.1 email_account

- The email account used to create an account against Let's Encrypt
- *type*: `string`
- *default*: `null` (no registration is done, and so no certificate is issued if an account does not exist yet)

3.3.2 staging

- If `true`, Let's Encrypt staging servers will be used (useful for testing purpose)
- *type*: `boolean`
- *default*: `false`

3.3.3 api_version

- The ACME protocol version to use (deprecated `1` or current `2`)
- *type*: `integer`
- *default*: `2`

3.3.4 directory_url

- The ACME CA server to use
- *type*: `string` representing a valid URL
- *default*: `null` (ACME CA server URL is determined using `staging` and `api_version` values)

3.3.5 `certs_permissions`

- An object describing the files and directories permissions to apply on generated certificates
- *type*: `object`
- *default*: `null` (default permissions are applied: certificates are owned by the user/group running DNSroboCert, and are only accessible by this user/group)

`files_mode`

- The permissions to apply to files, defined in POSIX octal notation
- *type*: `integer` or `string` defined in octal notation (eg. `0644` or `"0755"`)
- *default*: `0640`

`dirs_mode`

- The permissions to apply to directories, defined in POSIX octal notation
- *type*: `integer` or `string` defined in octal notation (eg. `0755` or `"0755"`)
- *default*: `0750`

`user`

- The user name or uid that should be owner of the certificates
- *type*: `integer` (for a uid, eg. `1000`) or `string` (for a user name, eg. `"myuser"`)
- *default*: `null` (user running DNSroboCert will be owner of the certificates)

`group`

- The group name or gid that should group owner of the certificates
- *type*: `integer` (for a gid, eg. `1000`) or `string` (for a user name, eg. `"mygroup"`)
- *default*: `null` (group running DNSroboCert will group owner of the certificates)

3.3.6 `crontab_renew`

- A cron pattern defining the frequency for certificates renewal check
- *type*: `string` representing a valid cron pattern
- *default*: `12 01,13 * * *` (twice a day)

3.4 `profiles` Section

This section holds *a list of profiles*. Each profile is an *object* that describes the credentials and specific configuration to apply to a DNS provider supported by Lexicon in order to fulfill a DNS-01 challenge.

Each profile is referenced by its `name`, which can be used in one or more certificates in the `certificates` section. Multiple profiles can be defined for the same DNS provider. However, each profile `name` must be unique.

```yaml
profiles:
  - name: my_profile1
    provider: digitalocean
    provider_options:
      auth_token: TOKEN
```

(continues on next page)

(continued from previous page)

```
    sleep_time: 45
    max_checks: 5
  - name: my_profile2_delegated
    provider: henet
    provider_options:
      auth_username: USER
      auth_password: PASSWORD
    delegated_subdomain: sub.example.net
```

3.4.1 `name`

- The name of the profile, used to reference this profile in the `certificates` section.
- *type*: `string`
- **mandatory property**

3.4.2 `provider`

- Name of the DNS provider supported by Lexicon
- *type*: `string`
- **mandatory property**

3.4.3 `provider_options`

- An *object* defining all properties to use for the DNS provider defined for this profile
- *type*: `object`
- *default*: `null`

Each property that should be added in `provider_option` depends on the actual provider used. You can check all properties available for each provider in the Lexicon Providers configuration reference page. As an example for Aliyun it will be:

```
provider_options:
  auth_key_id: MY_KEY_ID
  auth_secret: MY_SECRET
```

3.4.4 `sleep_time`

- Time in seconds to wait after the TXT entries are inserted into the DNS zone to perform the DNS-01 challenge of a certificate
- *type*: `integer`
- *default*: `30`

3.4.5 `max_checks`

- Maximum number of checks to verify that the TXT entries have been properly inserted into the DNS zone before performing the DNS-01 challenge of a certificate. DNSroboCert will wait for the amount of time defined in `sleep_time` between each check. Set to `0` to disable these checks.
- *type*: integer
- *default*: `0` (no check is done)

3.4.6 `delegated_subdomain`

- If the zone that should contain the TXT entries for the DNS-01 challenges is not a SLD (Second-Level Domain), for instance because a SLD delegated your subdomain to a specific zone, this options tells to DNSroboCert that your subdomain is actually the zone to modify, and not the SLD.
- For instance: the zone is `sub.example.net`, certificate is for `www.sub.example.net`, then `delegated_subdomain` should be equal to `sub.example.net`.
- *type*: `string`
- *default*: `null` (there is no subdomain delegation)

3.4.7 `ttl`

- Time to live in seconds for the TXT entries inserted in the DNS zone during a DNS-01 challenge.
- *type*: `integer`
- *default*: `null` (use any default TTL value specific to the DNS provider associated to this profile)

3.5 `certificates` Section

This sections handles the actual certificates that DNSroboCert needs to generate and renew regularly. It takes the form of **a list of certificates**. Each certificate is an object that describe the domains that needs to be included in the certificate, and the profile to use to handle the DNS-01 challenges: the profile is referred by its name, and **must** exist in the `profiles` Section.

In parallel several actions can be defined when a certificate is created or renewed. These actions have to be defined in each relevant certificate configuration.

```
certificates:
- name: my-wildcard-cert
  domains:
  - "*.example.net"
  - example.net
  profile: my_profile1
  pfx:
    export: true
    passphrase: PASSPHRASE
  autorestart:
  - containers:
    - container1
  - swarm_services:
    - service1
```

(continues on next page)

(continued from previous page)

```yaml
  podman_containers:
    - podman1
  autocmd:
  - cmd: /usr/bin/remote_deploy.sh
    containers:
    - container2
- domains:
  - www.sub.example.net
  profile: my_profile2_delegated
  deploy_hook: python /home/user/local_deploy.py
  force_renew: false
  follow_cnames: false
  reuse_key: false
  key_type: ecdsa
```

3.5.1 `profile`

- The profile name to use to validated DNS-01 challenges. This profile must exist in the `profiles` section.
- *type*: `string`
- **mandatory property**

3.5.2 `domains`

- List of the domains to include in the certificate.
- *type*: `list[string]`
- **mandatory property**

3.5.3 `name`

- Name of the certificate, used in particular to define where the certificate assets (key, cert, chain. . .) will be stored on the filesystem. For a certificate named `my-cert`, files will be available in the directory whose path is `[CERTS_PATH]/live/my-cert`. If the name is not specified, the effective certificate name will be the first domain listed in the `domains` property.
- *type*: `string`
- *default*: `null` (in this case name is extracted from the first domain listed in `domains`, for instance `example.net` for `example.net` or `*.example.net`)

3.5.4 `pfx`

- Configure an export of the certificate into the PFX (also known as PKCS#12) format upon creation/renewal.
- *type*: `object`
- *default*: `null` (certificate is not exported in PFX format)

`export`

- If *true*, the certificate is exported in PFX format.

- *type*: `boolean`
- *default*: `false` (the certificate is not exported in PFX format)

`passphrase`

- If set, the PFX file will be protected with the given passphrase.
- *type*: `string`
- *default*: `null` (the PFX file is not protected by a passphrase)

3.5.5 `deploy_hook`

- A command hook to execute locally when the certificate is created/renewed.
- *type*: `string`
- *default*: `null` (no deploy hook is configured)

Note: Several additional environment variables are injected by DNSrobocCert in the command runs by `deploy_hook`:

- `DNSROBOCERT_CERTIFICATE_NAME`: name of the current certificate in the configuration file,
- `DNSROBOCERT_CERTIFICATE_DOMAINS`: comma-separated list of the domains for the current certificate,
- `DNSROBOCERT_CERTIFICATE_PROFILE`: DNSroboCert profile associated with the current certificate.

3.5.6 `force_renew`

- If `true`, the certificate will be force renewed when DNSroboCert configuration changes. Useful for debugging purposes.
- *type*: `boolean`
- *default*: `false` (the certificate is not force renewed)

3.5.7 `follow_cnames`

- If `true`, DNSroboCert will follow the chain of CNAME that may be defined for the challenge DNS names `_acme-challenge.DOMAIN` (where `DOMAIN` is the domain to validate and integrate in the certificate). This allows to delegate the validation to another DNS zone for security purpose. See this link for more details.
- *type*: `boolean`
- *default*: `false` (CNAME chain is not followed)

3.5.8 `reuse_key`

- If `true`, the existing private key will be reused during certificate renewal instead of creating a new one each time the certificate is renewed.
- *type*: `boolean`

- *default*: `false` (the private key is never reused for certificate renewal)

3.5.9 `key_type`

- Type of key to use when the certificate is generated. Must be `rsa` or `ecdsa`.
- *type*: `string`
- *default*: `rsa` (a RSA-type key will be used)

Warning: The following paragraphs describe the `autorestart` and `autocmd` features. To allow them to work properly, DNSroboCert must have access to the Docker client socket file or the Podman socket. Usually at path:

- */var/run/docker.sock* for Docker,
- */run/podman/podman.sock* for rootful Podman,
- */run/user/$UID/podman/podman.sock* where $UID is your user id for rootless podman.

If DNSroboCert is run directly on the host, this usually requires to use a user with administrative privileges, or member of the *docker* group.

If DNSroboCert is run as a Docker, you will need to mount the Docker client socket file into the container. As an example the following command does that:

```
$ docker run --rm --name dnsrobocert
    --mount /var/run/docker.sock:/var/run/docker.sock
    adferrand/dnsrobocert
```

If DNSroboCert is run as a Podman, you will need to mount the podman socket into the container. As an example the following command does that:

For rootless Podman:

```
$ podman run --rm --name dnsrobocert
    --volume /run/user/$UID/podman/:/run/podman
    docker.io/adferrand/dnsrobocert
```

For rootful Podman:

```
$ sudo podman run --rm --name dnsrobocert
    --volume /run/podman/:/run/podman
    docker.io/adferrand/dnsrobocert
```

3.5.10 `autorestart`

- Configure an automated restart of target containers when the certificate is created/renewed. This property takes a list of autorestart configurations. Each autorestart is triggered in the order they have been inserted here.
- *type*: `list[object]`
- *default*: `null` (no automated restart is triggered)

`containers`

- A list of Docker containers to restart.
- *type*: `list[string]`

- *default*: `null` (no containers to restart)

`swarm_services`

- A list of swarm services to force restart
- *type*: `list[string]`
- *default*: `null` (no swarm services to restart)

`podman_containers`

- A list of Podman containers to restart.
- *type*: `list[string]`
- *default*: `null` (no containers to restart)

Property configuration example

```
autorestart:
- containers:
  - container1
  - container2
  swarm_services:
  - service1
```

3.5.11 `autocmd`

- Configure an automated execution of an arbitrary command on target containers when the certificate is is created/renewed. This property takes a list of autocmd configurations. Each autocmd is triggered in the order they have been inserted here.
- *type*: `list[object]`
- *default*: `null` (no automated command is triggered)

`cmd`

- The command to execute in each target container. Only commands of string type will be executed in a shell.
- *type*: `string` or `list[string]`
- **Mandatory property**

`containers`

- A list of Docker containers on which the command will be executed.
- *type*: `list[string]`
- *default*: `null` (no containers to restart)

Property configuration example

```
autocmd:
- containers:
  - container1
  - container2
  cmd: [echo, "Hello World!"]
- containers:
  - container3
  cmd: env
```

Warning: The feature `autocmd` is intended to call a simple executable file with few potential arguments. It is not made to call some advanced bash script, and would likely fail if you do so. In fact, the command is not executed in a shell on the target, and variables would be resolved against the DNSroboCert container environment. If you want to operate advanced scripting, put an executable script in the target container, and use its path in the relevant `autocmd[].cmd` property.

3.6 Environment variables

You can inject environment variables in the configuration file using the `${MY_VARIABLE}` format.

For instance, given that an environment variable named `AUTH_TOKEN` with the value `my-secret-token` exists, you can write the following file configuration content:

```yaml
profiles:
  - name: my_profile
    provider: digitalocean
    provider_options:
      auth_token: ${AUTH_TOKEN}
certificates: []
```

Then it will be resolved as:

```yaml
profiles:
  - name: my_profile
    provider: digitalocean
    provider_options:
      auth_token: my-secret-token
certificates: []
```

Non-existent variables declared in the configuration file will raise an error.

Note: If you want to write a literal `${NOT_A_VARIABLE}` that should not be resolved, you can escape the `${}` syntax by prepending a second dollar sign like so: `$${NOT_A_VARIABLE}`.

CHAPTER 4

Lexicon providers options

All Lexicon providers are supported by DNSroboCert.

You can look at the Lexicon's provider options page to get a list of the provider and their associated options.

CHAPTER 5

Miscellaneous

5.1 Activating staging ACME servers

During development it is not advised to generate certificates against production ACME servers, as one could reach easily the weekly limit of Let's Encrypt and could not generate certificates for a certain period of time. Staging ACME servers do not have this limit.

To use them, set the parameter `acme.staging` to `true` in your DNSroboCert YAML configuration file.

You will need to wipe content of /etc/letsencrypt volume before container re-creation when enabling or disabling staging. Otherwise accounts and/or certificates may be in conflict between their staging and production versions.

5.2 Executing the DNSroboCert docker in a specific timezone

The default timezone is UTC. You can set a local timezone in the docker *adferrand/dnsrobocert* by populating the `TIMEZONE` environment variable. In this case, automated renewal will be done in this timezone, and logs will use the local date.

5.3 Migration from docker-letsencrypt-dns

In this section we will discuss about how to migrate from `adferrand/letsencrypt-dns` to `adferrand/dnsrobocert`.

Indeed DNSroboCert started as a pure Docker implementation named `adferrand/letsencrypt-dns`. It was coded in bash, and was using both environment variables and a file named `domains.conf` for its configuration. `domains.conf` was holding the list of certificates to create and renew, and also the `autorestart` and `autocmd` features for each certificate. On the other hand, environment variables were configuring the DNS provider to use, the specific options for Let's Encrypt (account email address, staging servers) and some custom operations on the certificate assets (like specific users and permissions).

DNSroboCert supports all these features, improves them, and stores its configuration in one structured central file, located by default at `/etc/dnsrobocert/config.yml`. As said by DNSroboCert in the logs, usage of the old environment variables and the `domains.conf` file is deprecated, and **you should move as soon as possible to the `config.yml` file**. You should also use `adferrand/dnsrobocert` instead of `adferrand/letsencrypt-dns` starting from now.

If you followed the link displayed in logs from `adferrand/letsencrypt-dns`, then this section is for you: your instance of `letsencrypt-dns` has been upgraded to DNSroboCert, and you should migrate to `adferrand/dnsrobocert`.

Let's see this migration in details now.

5.3.1 Tool-assisted migration

Writing configuration files is boring. Do you agree? If so, you will be pleased to know that DNSroboCert handles this migration for you. Indeed if you start an `adferrand/dnsrobocert` instance with the legacy configuration (environment variables + `domains.conf`), DNSroboCert will automatically pick them and generate the new configuration file dynamically.

Its location is */etc/dnsrobocert/config-generated.yml*. It contains the necessary configuration to make DNSroboCert behave **exactly** like your `adferrand/docker-letsencrypt-dns` instance before.

Here are the steps to achieve the migration.

1. Pull the latest version of `adferrand/docker-letsencrypt-dns` and `adferrand/dnsrobocert`:

```
docker pull adferrand/docker-letsencrypt-dns
docker pull adferrand/docker-dnsrobocert
```

2. Restart your up-to-date instance of `adferrand/docker-letsencrypt-dns` appropriately with `docker` or `docker-compose` to ensure the new configuration file has been generated:

3. Extract this file from the docker into your host machine (assuming your docker is named `letsencrypt-dns`):

```
mkdir -p /etc/dnsrobocert
docker cp letsencrypt-dns:/etc/dnsrobocert/config-generated.yml /etc/dnsrobocert/
↪config.yml
```

4. Restart your Docker container with the new configuration file mounted at the right place:

   - With docker command line, add the following flag:

   ```
   --volume /etc/dnsrobocert/config.yml:/etc/dnsrobocert/config.yml
   ```

   - Or with docker-compose, add the mount directive in your `docker-compose.yml`

   ```
   volumes:
   - /etc/dnsrobocert/config.yml:/etc/dnsrobocert/config.yml
   ```

DNSroboCert will automatically pick the new configuration file.

5. Once you confirmed that everything is working as before, you can restart the Docker without the environment variables and `domains.conf` mount. Please take this occasion to change the image name from `adferrand/letsencrypt-dns` to `adferrand/dnsrobocert`. For instance:

```
docker run \
    --name dnsrobocert \
    --volume /var/docker-data/letsencrypt:/etc/letsencrypt \
```

(continues on next page)

(continued from previous page)

```
    --volume /etc/dnsrobocert/config.yml:/etc/dnsrobocert/config.yml \
    adferrand/dnsrobocert
```

Note: Docker image `adferrand/letsencrypt-dns` is deprecated and is replaced by `adferrand/dnsrobocert`.

5.3.2 Manual migration

If you want to go berserk, you can migrate yourself by writing the new `config.yml` file to fit your needs, following the documentation of the User guide and Configuration reference.

Once done, you can follow the previous section to restart your Docker container.

5.3.3 Former configuration of `adferrand/letsencrypt-dns`

If needed, the former configuration for `adferrand/letsencrypt-dns` is available on GiHub.

5.3.4 What is new?

At this point, you may ask yourself what you gain by migrating from `adferrand/letsencrypt-dns` to `adferrand/dnsrobocert`.

Well, thanks to this migration a lot of new features are planned, since this is a complete refactoring of the tool into a proper programming language, Python. Basically it becames a real program that I name DNSroboCert, with code quality control and good extensibility to add all the features the community asks for.

You can check in particular the Project V3 specifications that drove this migration and gives key points for the incoming features.

But beyond promises you will get immediate advantages that I already implemented in DNSroboCert:

- **the big one**: you can now define multiple DNS providers in one single instance of DNSroboCert
- the custom deploy scripts and PFX exports are defined per certificate
- force renew can be set for specific certificates

Stay tuned for the new features!

CHAPTER 6

Developer guide

Thanks a lot for your involvement. You will find here some tips to quickly setup a suitable development environment, and useful tools to ensure code quality in the project.

6.1 General design principles

DNSroboCert is coded entirely in Python, and uses features available starting with Python 3.6+.

It sits on top of Certbot and Lexicon. Here are the repartition of the roles:

- Certbot takes care of the actual certificate issuances and renewals against the ACME CA server, in a compliant and secured processing that respects the RFC-8555 protocol,
- Lexicon provides the central interface to communicate with the DNS API providers, and inserts the required TXT entries for the DNS-01 challenges,
- DNSroboCert
  - holds and validates the central configuration for users,
  - couples Certbot and Lexicon through the auth/cleanup hook system of Certbot's manual plugin, to issue/renew DNS-01 challenged based certificates,
  - orchestrates the post-deploy processing (`autocmd`, `autorestart`, files rights…),
  - executes a a cron job to trigger regularly the Certbot renewal process.

6.2 Setting up a development environment

DNSroboCert uses Poetry to configure an environment development, build the project, and push wheel/sdist to PyPI.

1. First, install Poetry, following this guide: Poetry installation.
2. Now Poetry should be available in your command line. Check that the following command is displaying Poetry version:

```
poetry --version
```

3. Fork the upstream GitHub project and clone your fork locally:

```
git clone https://github.com/myfork/dnsrobocert.git
```

Note:

A widely used development pattern in Python is to setup a virtual environment.

Python virtual environments allow to manage a dedicated and isolated Python runtime for a specific project.

It allows in particular to have a separated set of dependencies for the project that will not interfere with the OS Python packages installed globally.

4. Setup the virtual environment for DNSroboCert using Poetry:

```
cd dnsrobocert
poetry env use python3
```

5. Activate the virtual environment:

- For Linux/Mac OS X:

```
source .venv/bin/activate
```

- For Windows (using Powershell):

```
.\.venv\Scripts\activate
```

6. Install development dependencies.

```
poetry install
```

At this point, you are ready to develop on the project. You can run the CLI that will use the local source code:

```
dnsrobocert --help
```

6.3 Code quality

The project DNSroboCert tries to follows the up-to-date recommended guideline in Python development:

- DNSroboCert logic is tested with a pyramidal approach (unit tests + integration tests) using Pytest.
- The code is formatted using Black and Isort to keep as possible unified and standardized writing conventions.
- The code is linted with Flake8 and statically checked using MyPy.

Please ensure that your code is compliant with this guideline before submitting a PR:

1. Ensure that tests are passing:

```
pytest test
```

Warning: On Windows you must run the tests from an account with administrative privileges to make them pass.

2. Ensure that linting and static type checking are passing:

```
flake8 src test utils
mypy src
```

3. Reformat your code:

```
isort -rc src test utils
black src test utils
```

6.4 Submitting a PR

Well, you know what to do ;)