Certhub Documentation

Lorenz Schori

Apr 03, 2022

Contents:

CHAPTER 1

Intro

Thanks to Let's Encrypt and Certbot the number of TLS enabled sites is growing faster than ever. Transport encryption is becoming ubiquitous leading to much better security on the user facing web.

Many of the available ACME clients (including Certbot) made it a top priority to very well support monolithic systems which are managed manually. As a result ACME client software commonly assumes that it has access to TLS private keys, privileged ports or the web servers configuration files in order to make the process of obtaining a certificate as smooth as possible.

Admittedly this simplifies the life of inexperienced system administrators. But the lack of proper privilege separation also poses a risk to the systems integrity.

Let's Encrypt certificates have a very limited life-span of 90 days. Therefore it is crucial to automate certificate renewal as well. Most ACME clients employ some mechanism to track the state of issued certificates and trigger a timely renewal. In order to allow for rollbacks, ACME clients often archive previous versions of certificates in some directory structure.

As a result some state needs to be backed up and preseeded when a system gets rebuilt. With the rise of IT automation, containers and virtual machines are built and deployed more frequently. Keeping software state across those builds in various vendor specific formats is inconvenient.

1.1 Configuration

Certhub strictly separates (read-only) configuration from (read-write) state. As a result it naturally integrates well with configuration management and IT automation systems.

1.2 State

Instead of maintaining an ad-hoc directory structure of current and previous certificates, Certhub keeps track of them using a git repository.

Certhub provides a simple utility to check for certificates which are about to expire. The predefined action is to trigger the certificate renewal job if an outdated certificate is detected. Both, the expiry check and the renewal job do operate on temporary checkouts of the git repository.

1.3 Replication

Keeping certificates in a repository also makes it easier to replicate fresh certificates to other machines or make them available for new deployments on a repository hosting service.

There is a large body of CI/CD systems which integrate well with git. With Certhub a certificate renewal results in a new commit to the repository. Thus existing automation infrastructure can be leveraged to trigger builds and deployments without hooking into ACME client software directly.

1.4 Separation

Granted that Certhub takes over state management, repository replication and expiry checks, the only thing left for the actual ACME client is to obtain certificates.

Some ACME clients support an operation mode where access to the TLS private key is not necessary. A CSR is taken as the input and the resulting certificate is written to an output file. This operation can be performed without elevated privileges. Certhub currently supports Certbot, Dehydrated and Lego in unprivileged CSR mode.

Note that this mode of operation simplifies centralized certificate management where the ACME client is only installed on one or few machines separated from the systems which are running the actual TLS servers.

CHAPTER 2

Overview

2.1 System architecture

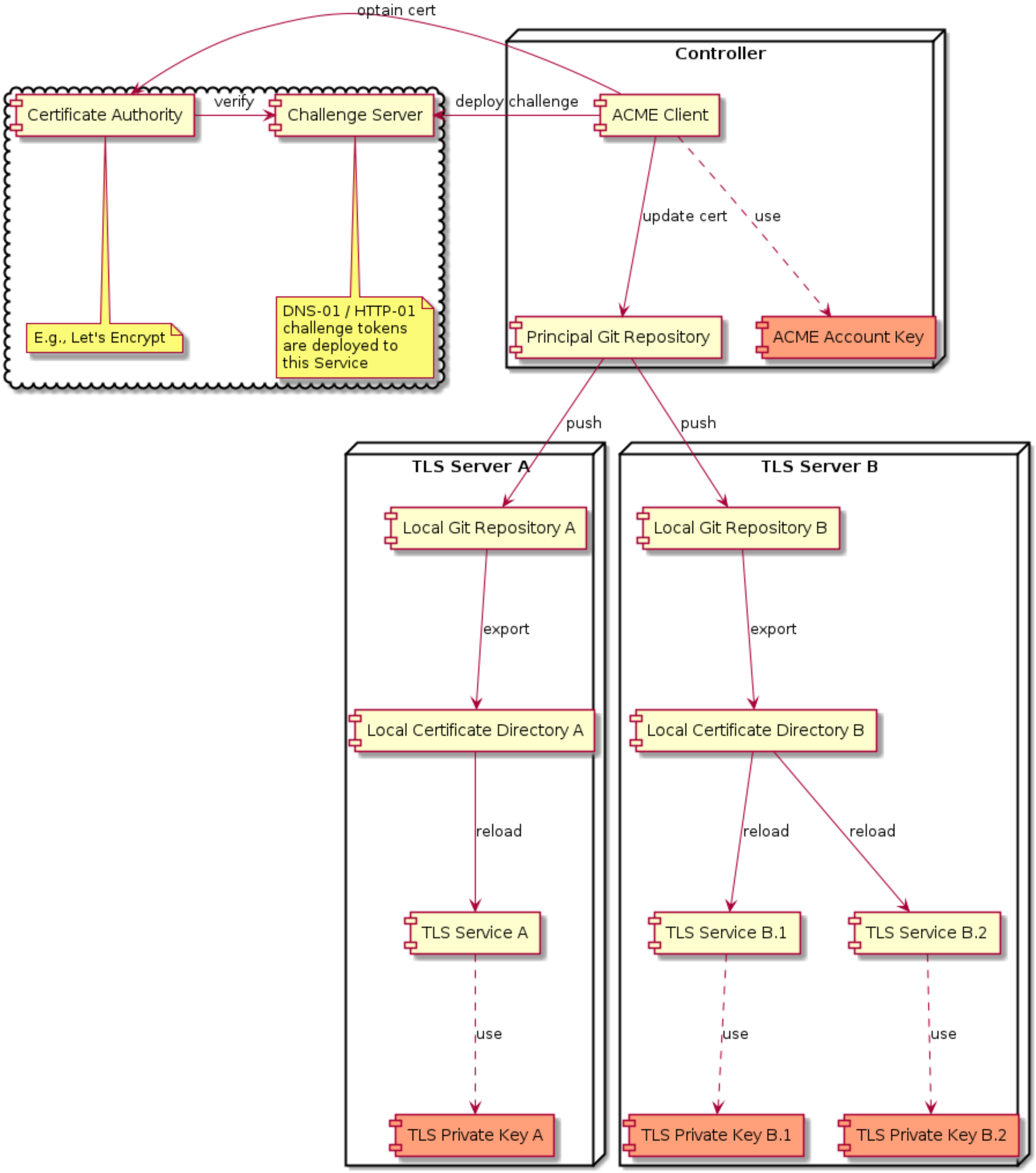
A typical certhub deployment consists of one *Controller* node hosting the *ACME Client* (i.e., `Certbot`, `Dehydrated` or `Lego`) along with the *Principal Git Repository*. Multiple *TLS Server* nodes are used to host *TLS Services* such as web servers, mail servers, databases and application servers.

The *ACME Client* stores certificates optained from the *Certificate Authority* in the *Principal Git Repository*. Changes to that repo are replicated to the *Local Git Repositories* on the *TLS Server* nodes automatically.

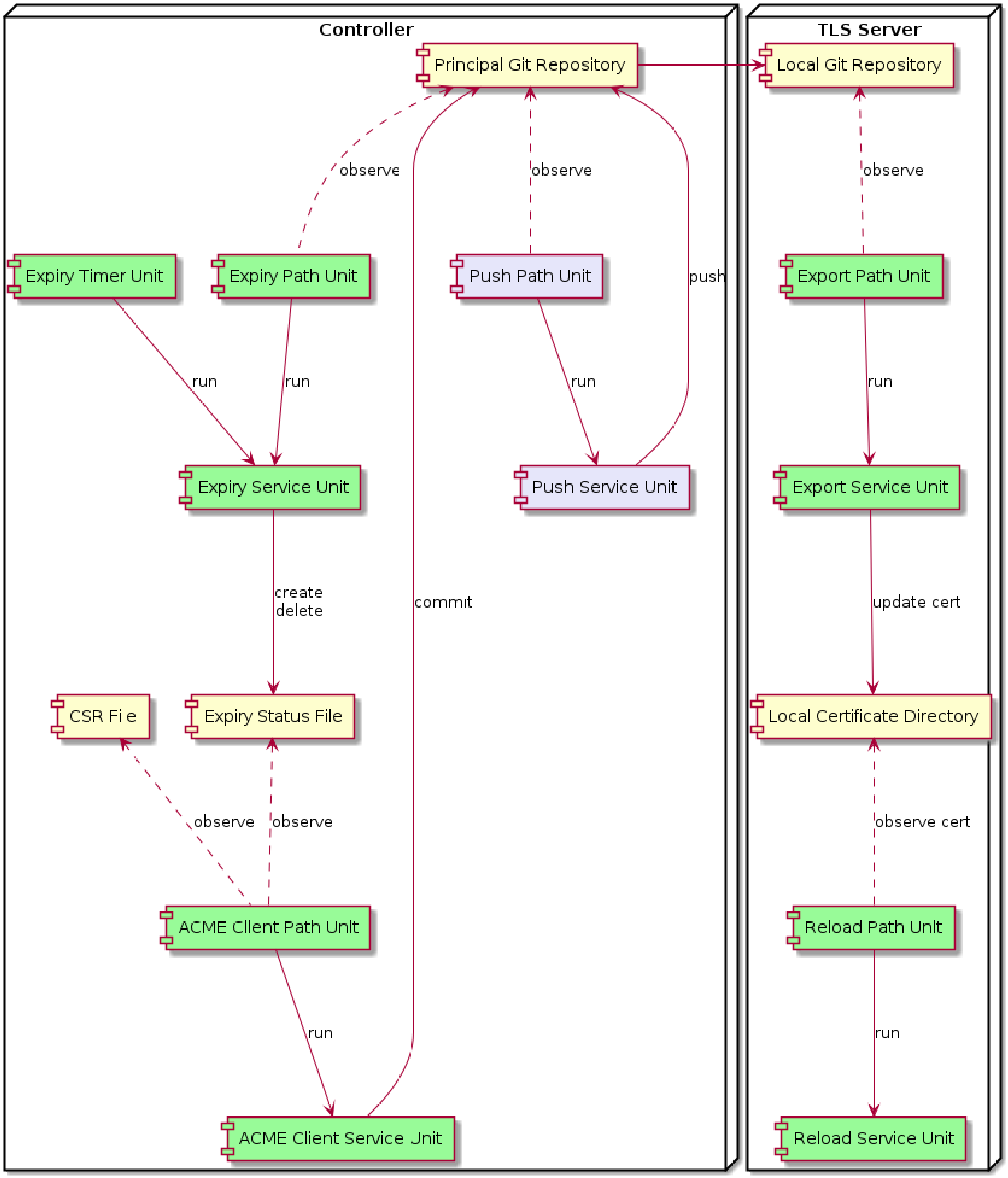
Certificates in the *Local Certificate Directory* on a *TLS Server* node are updated automatically whenever new commits are pushed to the *Local Git Repository*.

TLS Services are reloaded whenever the exported certificate in the *Local Certificate Directory* is modified.

Note that *ACME Account Key* is only needed on the *Controller* node (readable by the *ACME Client*). Also *TLS Private Keys* are only deployed on the respective *TLS Server* nodes and are readable by their respective *TLS services* exclusively.



Certhub ships with a considerable number of systemd units. All of those are designed as templates. Units used for replication to other nodes use the destination for the template instance string, all other units take a certificate basename as their instance string. The following diagram depicts a typical setup featuring automatic renewal, replication, certificate export and TLS service reload.



2.2 Controller node setup process

In a typical certhub setup there is only **one** *Controller* node. Setting up the *Controller* isn't something which is repeated frequently.

In order to setup a new *Controller* node, the following steps are required. For production deployments it is recommended to use a configuration management system.

On the *Controller* node:

1. Install required software including certhub and its dependencies. Also install one of the supported *ACME client*.
2. Setup the local unprivileged certhub user account.
3. Generate an SSH keypair to be used for repository replication.
4. Initialize the *Principal Git Repository*.
5. Create the necessary directory structure including private directory for *ACME Account Keys* as well as config and state directories.
6. Create or restore the *ACME Account Keys* for the installed *ACME Client*.

2.3 TLS Server node setup process

In a typical certhub setup there are more than one *TLS Server* node. Depending on the environment, *TLS Server* nodes might get deployed regularely.

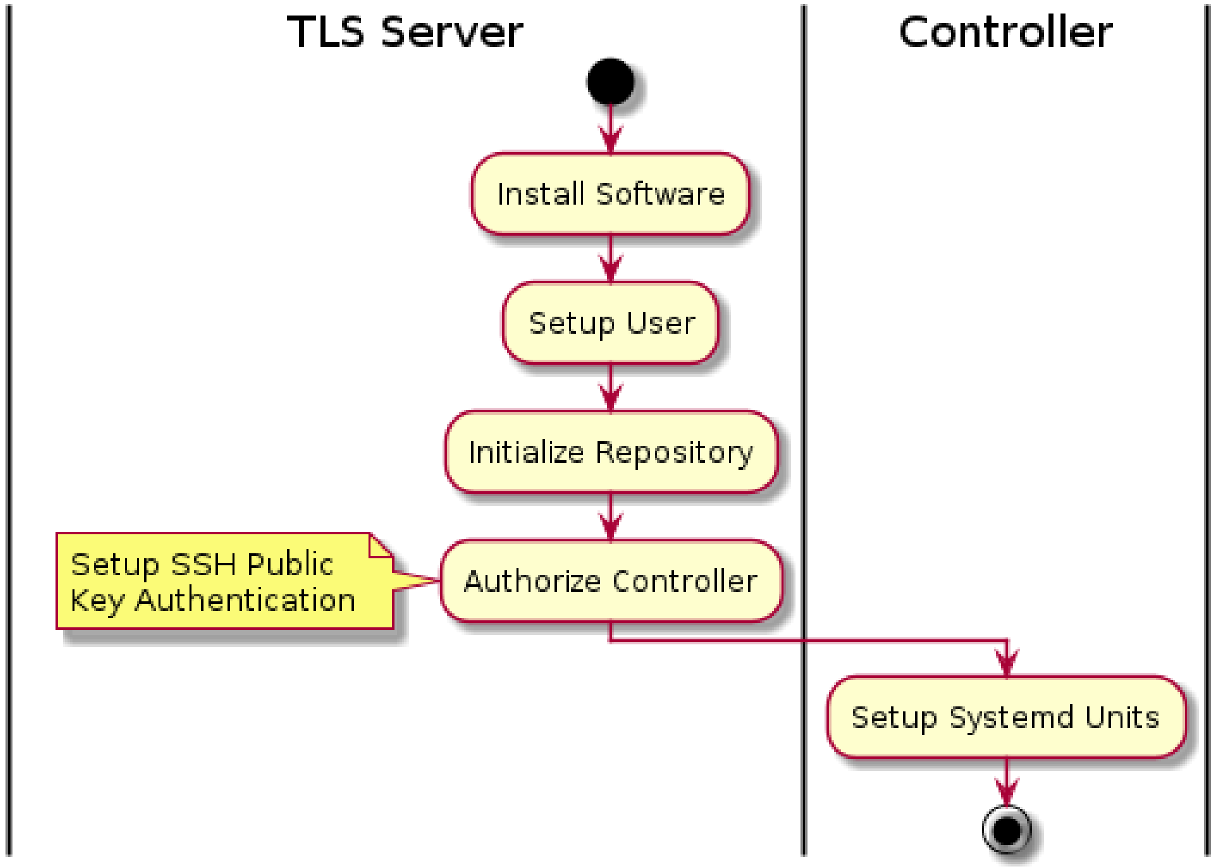
In order to setup a new *TLS Server* node, the following steps are required. For production deployments it is recommended to use a configuration management system.

On the *TLS Server* node:

1. Install required software including certhub and its dependencies. Do **not** install any *ACME client* software on *TLS Server* nodes.
2. Setup the local unprivileged certhub user account.
3. Initialize the *Local Git Repository* and create the *Local Certificate Directory*.
4. Authorize the certhub user on the *Controller* node to push to the *Local Git Repository*.

On the *Controller* node:

1. Setup systemd units responsible for replicating the *Principal Git Repository* to the *Local Git Repository* on the new *TLS Server*.



2.4 TLS Service setup process

In a typical certhub setup there are more than one *TLS Service*. Depending on the environment, *TLS Services* might get deployed regularely.

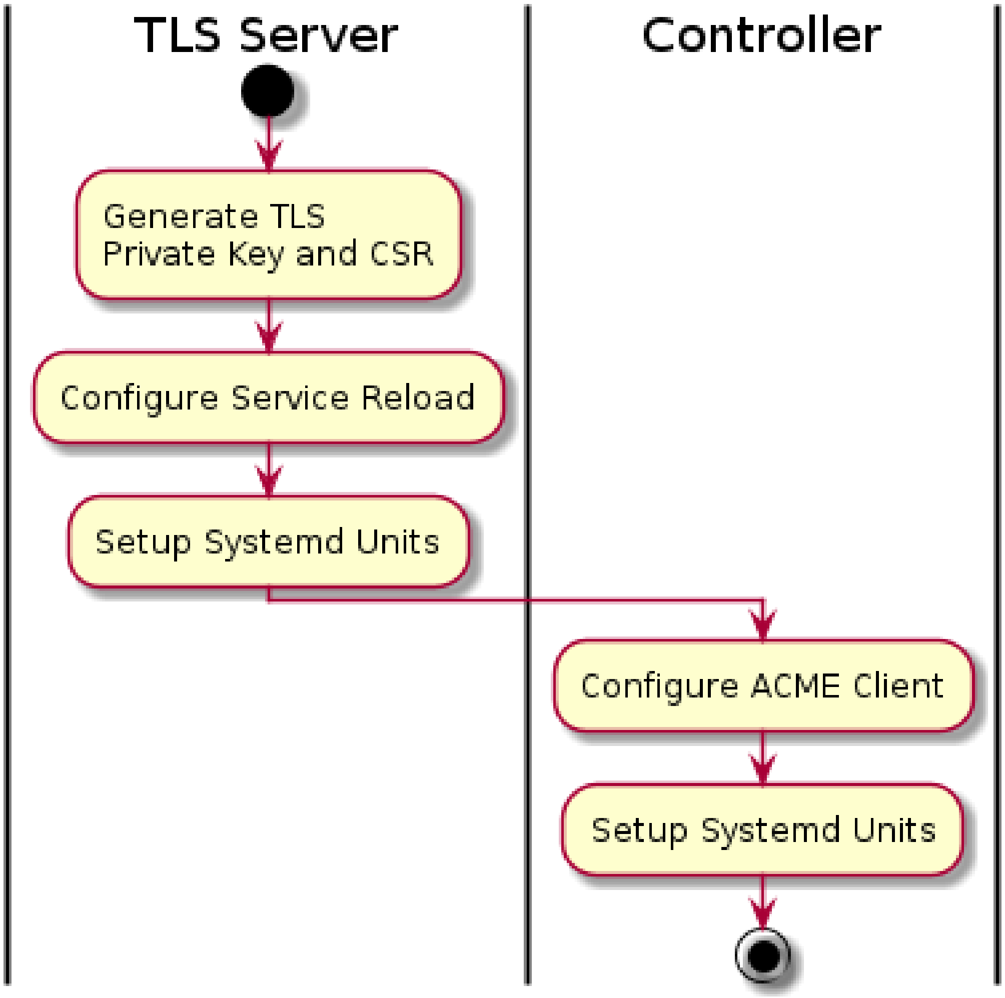
The following steps are needed to create a new certificate for a new *TLS Service*. For production deployments it is recommended to use a configuration management system.

On the *TLS Server* node:

1. Generate a new *TLS Private Key* and *Certificate Signing Request* (*CSCR*).
2. Add a configuration file which specifies the *TLS Service(s)* to be reloaded whenever the certificate changes in the *Local certificate Directory*.
3. Setup systemd units responsible for exporting changed certificates and reloading services.

On the *Controller* node:

1. Add the newly generated *CSR* along with *ACME Client* specific configuration to the certhub config directory.
2. Setup systemd units responsible for checking certificate expiry and automatic renewal.


TLS Server
Controller
Generate TLS Private Key and CSR
Configure Service Reload
Setup Systemd Units
Configure ACME Client
Setup Systemd Units

CHAPTER 3

Installation

3.1 Dependencies

The executables provided by certhub only depend on `openssl` and any of the following supported ACME clients: Certbot, Dehydrated or Lego. Certhub includes DNS-01 challenge hooks for `nsupdate` and Lexicon.

In order to use the systemd units, `git` and git-gau is required.

3.2 Install

Navigate to the Certhub releases page and pick the latest `certhub-dist.tar.gz` tarball. Copy it to the target machine and unpack it there.

```
$ scp dist/certhub-dist.tar.gz me@example.com:~
$ ssh me@example.com sudo tar -C /usr/local -xzf ~/certhub-dist.tar.gz
```

Alternatively use the following ansible task to copy and unarchive a dist tarball into */usr/local*. Note that git-gau can be installed in the same way.

```
- name: Certhub present
  notify: Systemd reloaded
  unarchive:
    src: files/certhub-dist.tar.gz
    dest: /usr/local
```

CHAPTER 4

Systemd Setup

Certhub ships with a host of systemd `service`, `timer` and `path` units which can be combined in various ways to satisfy different use cases.

Most of them are designed to run as an unprivileged system user. The systemd units default to `certhub` as the UID as well as the primary GID. Also the home directory by default is expected to be located at `/var/lib/certhub`.

4.1 Certhub User

Add the `certhub` user and group on the target system. Use the `--shell /usr/bin/git-shell` option in order to enable `git` repository replication.

```
$ sudo adduser --system --group --home /var/lib/certhub --shell /usr/bin/git-shell␣
↪certhub
```

Same thing as an Ansible task:

```
- name: Certhub user present
  user:
    name: certhub
    state: present
    system: yes
    home: /var/lib/certhub
    shell: /usr/bin/git-shell
```

4.2 Directory Structure

Configuration for each certificate is expected in `/etc/certhub`. This directory must not be writable by the `certhub` user. Any files holding secrets such as API tokens or DNS TSIG keys should be owned by `root` with group `certhub` and permissions `0640` in order to prevent leaking information to other unprivileged processes.

```
$ sudo mkdir /etc/certhub
```

```
- name: Certhub config directory present
  file:
    path: /etc/certhub
    state: directory
    owner: root
    group: root
    mode: 0755
```

A status directory is used to trigger certificate renewal via systemd `path` units. The `certhub` user obviously needs write access to that directory. By default it is located at `/var/lib/certhub/status`.

```
$ sudo -u certhub mkdir /var/lib/certhub/status
```

```
- name: Certhub status directory present
  file:
    path: /var/lib/certhub/status
    state: directory
    owner: certhub
    group: certhub
    mode: 0755
```

4.3 Local Repository

By default systemd units expect the local certificate repository in `/var/lib/certhub/certs.git`. Also it is recommended to at least set `user.name` and `user.email` in the `git` configuration of the `certhub` user. Some `git` versions complain if `push.default` is not set. Thus it is best to specify that explicitly as well.

```
$ sudo -u certhub git config --global user.name Certhub
$ sudo -u certhub git config --global user.email certhub@$(hostname -f)
$ sudo -u certhub git config --global push.default simple
$ sudo -u certhub git init --bare /var/lib/certhub/certs.git
$ sudo -u certhub git gau-exec /var/lib/certhub/certs.git git commit --allow-empty -m
↪'Init'
```

```
- name: Git configured
  become: yes
  become_user: certhub
  loop:
    - { name: "user.name", value: Certhub }
    - { name: "user.email", value: "certhub@{{ ansible_fqdn }}" }
    - { name: "push.default", value: simple}
  git_config:
    name: "{{ item.name }}"
    value: "{{ item.value }}"
    scope: global

- name: Certhub repository present
  become: yes
  become_user: certhub
  command: >
    git init --bare /var/lib/certhub/certs.git
```

(continues on next page)

(continued from previous page)

```
  arg:
    creates: /var/lib/certhub/certs.git

- name: Certhub repository initialized
  become: yes
  become_user: certhub
  command: >
    git gau-exec /home/certhub/certs.git
    git commit --allow-empty -m'Init'
  arg:
    creates: /var/lib/certhub/certs.git/refs/heads/master
```

4.4 ACME Client Setup

ACME clients need a way to store Let's Encrypt account keys. By default systemd units expect home directories for the supported ACME clients to be inside `/var/lib/certhub/private/`. This directory must not be world-readable.

```
$ sudo -u certhub mkdir -m 0700 /var/lib/certhub/private
```

```
- name: Certhub private directory present
  file:
    path: /var/lib/certhub/private
    state: directory
    owner: certhub
    group: certhub
    mode: 0700
```

4.4.1 Certbot Setup

Certbot needs some special configuration in order to make it run as an unprivileged user. The following configuration directives need to be placed inside `/var/lib/certhub/.config/letsencrypt/cli.ini`.

Also the referenced directories should be created before running certbot for the first time.

Shell:

```
$ sudo -u certhub mkdir -p /var/lib/certhub/private/certbot/{work,config,logs}
$ sudo -u certhub mkdir -p /var/lib/certhub/.config/letsencrypt
$ sudo -u certhub tee /var/lib/certhub/.config/letsencrypt/cli.ini <<EOF
work-dir = /var/lib/certhub/private/certbot/work
config-dir = /var/lib/certhub/private/certbot/config
logs-dir = /var/lib/certhub/private/certbot/logs
EOF
```

Ansible:

```
- name: Certbot directory structure present
  loop:
    - /var/lib/certhub/.config/letsencrypt
    - /var/lib/certhub/private/certhub/work
    - /var/lib/certhub/private/certhub/config
```

(continues on next page)

(continued from previous page)

```
    - /var/lib/certhub/private/certhub/log
  file:
    path: "{{ item }}"
    state: directory
    recursive: true
    owner: certhub
    group: certhub
    mode: 0755

- name: Certbot cli.ini present
  copy:
    dest: /var/lib/certhub/.config/letsencrypt/cli.ini
    owner: certhub
    group: certhub
    mode: 0755
    content: |
      work-dir = /var/lib/certhub/private/certbot/work
      config-dir = /var/lib/certhub/private/certbot/config
      logs-dir = /var/lib/certhub/private/certbot/logs
```

4.4.2 Dehydrated Setup

Shell:

```
$ sudo -u certhub mkdir /var/lib/certhub/private/dehydrated
```

Ansible:

```
- name: Dehydrated directory present
  file:
    path: /var/lib/certhub/private/dehydrated
    state: directory
    owner: certhub
    group: certhub
    mode: 0755
```

4.4.3 Lego Setup

Shell:

```
$ sudo -u certhub mkdir -p /var/lib/certhub/private/lego/{accounts,certificates}
```

Ansible:

```
- name: Lego directory structure present
  loop:
    - /var/lib/certhub/private/lego/accounts
    - /var/lib/certhub/private/lego/certificates
  file:
    path: "{{ item }}"
    state: directory
    recursive: true
    owner: certhub
```

(continues on next page)

(continued from previous page)

```
    group: certhub
    mode: 0755
```

4.4.4 Domain Validation

Choose the challenge method which best suits the infrastructure. DNS-01 challenge is unavoidable for wildcard certificates. Currently DNS-01 is the only method which is supported out-of-the box by certhub and which is covered by integration tests.

Certhub ships with DNS-01 challenge hooks for `nsupdate` and Lexicon. The hooks need to be configured using an environment file normally located in *`/etc/certhub/%i.certhub-certbot-run.env`* and *`/etc/certhub/%i.certhub-dehydrated-run.env`*. An example for certbot and dehydrated configuration is part of the integration test suite. See the manpages *certhub-hook-lexicon-auth* and *certhub-hook-nsupdate-auth* for more detailed information about the involved environment variables.

In the case of lego the challenge method is selected using command line arguments to the lego binary, authentication tokens are passed in via environment variables. All configuration is passed in via an environment file normally located in *`/etc/certhub/%i.certhub-lego-run.env`*. An example configuration is part of the integration test suite. See the manpage *certhub-lego-run@.service* for more detailed information about the involved environment variables.

Note that it is not recommended to specify secrets like API tokens in environment variables or command line flags. Regrettably most of today's software authors seem to ignore this fact. An effective way to prevent secrets from leaking via process table is to keep them in files with tight access restrictions. Regrettably neither Lexicon nor lego do support this approach. Thus for production grade setups it is unavoidable to either use the `nsupdate` method or implement custom challenge hook scripts which are capable of reading API tokens from files.

Also note that HTTP-01 validation can be implemented quite easily if a reverse proxy serving the whole range of sites is already in place. In this case it is enough to proxy the path `.well-known/acme-challenge` to the certhub controller and then run a HTTP server and an ACME client in webroot-mode.

Refer to the following section for detailed directives on how to customize services via drop-ins.

4.5 Systemd Unit Customization

Certhub ships with systemd units which are capable of running one of the supported ACME clients in order to issue or renew a certificate and then store it in the certificate repository.

All the units are extensively configurable via systemd unit drop-ins. Units and drop-ins shipped with certhub are located in `lib/systemd/system` inside the installation prefix (usually `/usr/local`). Create corresponding drop-in directories inside `/etc/systemd/system` and then copy over selected drop-ins in order to customize certhub `service`, `path` and `timer` units.

4.6 Certificates

All systemd units are designed as templates. The instance name serves as the basename for configuration as well as generated certificates.

In order to avoid problems it is recommended to only use characters allowed in path components. I.e., alphanumeric plus URL-safe special characters such as the period and minus.

The following steps are required to configure a new certificate.

4.6.1 CSR

Generate a CSR from the TLS servers private key. When working with Ansible use delegation to run the `openssl req` command on another host than the certhub controller. Add the CSR to `/etc/certhub/${DOMAIN}.csr.pem`. In simple setups it is recommended to use the domain name as the config base name.

Shell:

```
$ export SERVER=tls-server.example.com
$ export DOMAIN=tls-server.example.com
$ ssh "${SERVER}" sudo openssl req -new \
    -key "/etc/ssl/private/${DOMAIN}.key.pem" \
    -subj "/CN=${DOMAIN}" \
    | sudo tee "/etc/certhub/${DOMAIN}.csr.pem"
```

Ansible:

```
- name: CSR generated
  delegate_to: "{{ SERVER }}"
  changed_when: false
  register: csr_generated
  command: >
    openssl req -new
    -key "/etc/ssl/private/{{ DOMAIN }}.key.pem"
    -subj "/CN={{ DOMAIN }}"

- name: CSR configured
  register: csr_configured
  copy:
    dest: "/etc/certhub/{{ DOMAIN }}.csr.pem"
    content: "{{ csr_generated.stdout }}
    owner: root
    group: root
    mode: 0644
```

4.6.2 ACME Client Configuration

Add additional configuration for the ACME client to one of the following files: `/etc/certhub/${DOMAIN}.certbot.ini`, `/etc/certhub/${DOMAIN}.dehydrated.conf` or `/etc/certhub/${DOMAIN}.certhub-lego-run.env`. Working examples for testing purposes are part of certhub integration tests

4.6.3 Initial Certificate

Run `certhub-${ACME_CLIENT}-run@${DOMAIN}.service` once in order to obtain the first certificate and add it to the repository.

Example for `ACME_CLIENT=certbot` and `DOMAIN=tls-server.example.com`

```
$ export ACME_CLIENT=certbot
$ export DOMAIN=tls-server.example.com
$ sudo systemctl start "certhub-${ACME_CLIENT}-run@${DOMAIN}.service"
```

Ansible:

```
- name: Certificate issued
  systemd:
    name: "certhub-{{ ACME_CLIENT }}-run@{{ DOMAIN }}.service"
    state: started
```

4.6.4 Configure Certificate Renewal

Enable and start `timer` and `path` units.

Shell:

```
$ export DOMAIN=tls-server.example.com
$ sudo systemctl enable --now "certhub-cert-expiry@${DOMAIN}.path"
$ sudo systemctl enable --now "certhub-cert-expiry@${DOMAIN}.timer"
$ sudo systemctl enable --now "certhub-certbot-run@${DOMAIN}.path"
```

Ansible:

```
- name: Path and timer units enabled and started
  loop:
    - "certhub-cert-expiry@{{ DOMAIN }}.path"
    - "certhub-cert-expiry@{{ DOMAIN }}.timer"
    - "certhub-certbot-run@{{ DOMAIN }}.path"
  systemd:
    name: "{{ item }}"
    enabled: true
    state: started
```

4.7 Certificate Distribution

In order to propagate certificates to tls servers it is recommended to mirror the repository from the certhub controller to the respective machines. The `certhub-repo-push@.service` unit can be used to propagate these changes to another host, `certhub-repo-push@.path` unit to trigger it automatically whenever the `master` branch of the repository changes.

Note, `certhub-repo-push@.service` requires working SSH access via public key authentication to the remote end.

This unit takes the full remote URL including the path as the service instance name which needs to be escaped using `systemd-escape --template`. Note, when copy-pasting output from `system-escape` into a shell then it is necessary to escape backslashes with an additional backslash.

Shell:

```
$ export REMOTE="tls-server.example.com:/var/lib/certhub/certs.git"
$ export PATH_UNIT="$(systemd-escape --template certhub-repo-push@.path ${REMOTE})"
$ export SERVICE_UNIT="$(systemd-escape --template certhub-repo-push@.service $
↪{REMOTE})"
$ sudo systemctl enable --now "${PATH_UNIT}"
$ sudo systemctl start "${SERVICE_UNIT}"
```

Ansible:

```
tasks:
  - name: Certificate distribution activated
    notify: Certificate distribution run
    vars:
      UNIT: "{{ lookup('pipe','systemd-escape --template certhub-repo-push@.path ' +␣
↪REMOTE|quote) }}"
    systemd:
      name: "{{ UNIT }}"
      enabled: true
      state: started

handlers:
  - name: Certificate distribution run
    vars:
      UNIT: "{{ lookup('pipe','systemd-escape --template certhub-repo-push@.service '␣
↪+ REMOTE|quote) }}"
    systemd:
      name: "{{ UNIT }}"
      state: started
```

4.8 Certificate export and service reload

Whenever a new commit is pushed to the local repository on a tls server node, selected certificates may be exported such that they can be used in the config of tls servers. Also affected tls services should be reloaded wenever an exported certificate was renewed. Enable and start `certhub-cert-export@.path` and `certhub-cert-reload@.path` in order to automate this process on tls server nodes. Both of these units take a certificate configuration basename as their instance name.

All units which should be reloaded whenever the exported certificate changes should be listed in `/etc/certhub/${DOMAIN}.services-reload.txt`.

The default destination for exported certificates is `/var/lib/certhub/certs`.

Shell:

```
$ export DOMAIN=tls-server.example.com
$ sudo -u certhub mkdir /var/lib/certhub/certs
$ sudo tee "/etc/certhub/${DOMAIN}.services-reload.txt" <<EOF
nginx.service
EOF
$ sudo systemctl enable --now "certhub-cert-export@${DOMAIN}.path"
$ sudo systemctl enable --now "certhub-cert-reload@${DOMAIN}.path"
$ sudo systemctl start "certhub-cert-export@${DOMAIN}.service"
```

Ansible:

```
tasks:
  - name: Certhub certificate directory exists
    file:
      path: /var/lib/certhub/certs
      state: directory
      owner: certhub
      group: certhub
      mode: 0755

```

(continues on next page)

(continued from previous page)

```
  - name: Service reload configuration
    copy:
      dest: "/etc/certhub/{{ DOMAIN }}.services-reload.txt"
      owner: root
      group: root
      mode: 0644
      content: |
        nginx.service

  - name: Certificate export and service reload path units enabled and started
    notify: Certificate exported
    loop:
      - "certhub-cert-export@{{ DOMAIN }}.path"
      - "certhub-cert-reload@{{ DOMAIN }}.path"
    systemd:
      name: "{{ item }}"
      enabled: true
      state: started

handlers:
  - name: Certificate exported
    systemd:
      name: "certhub-cert-export@{{ DOMAIN }}.service"
      state: started
```

4.9 Sending certificates

Similar to the export/reload scenario described above, it is also possible to send exported certificates to another destination/process. Enable and start `certhub-cert-export@.path` and `certhub-cert-send@.path` in order to automate this process. Both of these units take a certificate configuration basename as their instance name.

List all destinations the certificate should be sent to in `/etc/certhub/${DOMAIN}.destinations-send.txt`. By default the certificate will be sent using the `mail` command. This can be changed using the *`CERTHUB_CERT_SEND_COMMAND`*. A good place to specify the variable is, e.g., *`/etc/certhub/%i.certhub-cert-send.env`*.

Note that the certificate is written to `stdin` of the specified command. Hence it is quite easy to send it to remote scripts using `ssh`.

Shell:

```
$ export DOMAIN=tls-server.example.com
$ sudo -u certhub mkdir /var/lib/certhub/certs
$ sudo tee "/etc/certhub/${DOMAIN}.destinations-send.txt" <<EOF
audit@example.com
EOF
$ sudo systemctl enable --now "certhub-cert-export@${DOMAIN}.path"
$ sudo systemctl enable --now "certhub-cert-send@${DOMAIN}.path"
$ sudo systemctl start "certhub-cert-export@${DOMAIN}.service"
```

Ansible:

```
tasks:
  - name: Certhub certificate directory exists
    file:
```

(continues on next page)

(continued from previous page)

```
      path: /var/lib/certhub/certs
      state: directory
      owner: certhub
      group: certhub
      mode: 0755

  - name: Certificate send configuration
    copy:
      dest: "/etc/certhub/{{ DOMAIN }}.destinations-send.txt"
      owner: root
      group: root
      mode: 0644
      content: |
        audit@example.com

  - name: Certificate export and send path units enabled and started
    notify: Certificate exported
    loop:
      - "certhub-cert-export@{{ DOMAIN }}.path"
      - "certhub-cert-send@{{ DOMAIN }}.path"
    systemd:
      name: "{{ item }}"
      enabled: true
      state: started

handlers:
  - name: Certificate exported
    systemd:
      name: "certhub-cert-export@{{ DOMAIN }}.service"
      state: started
```

CHAPTER 5

GitLab CI Setup

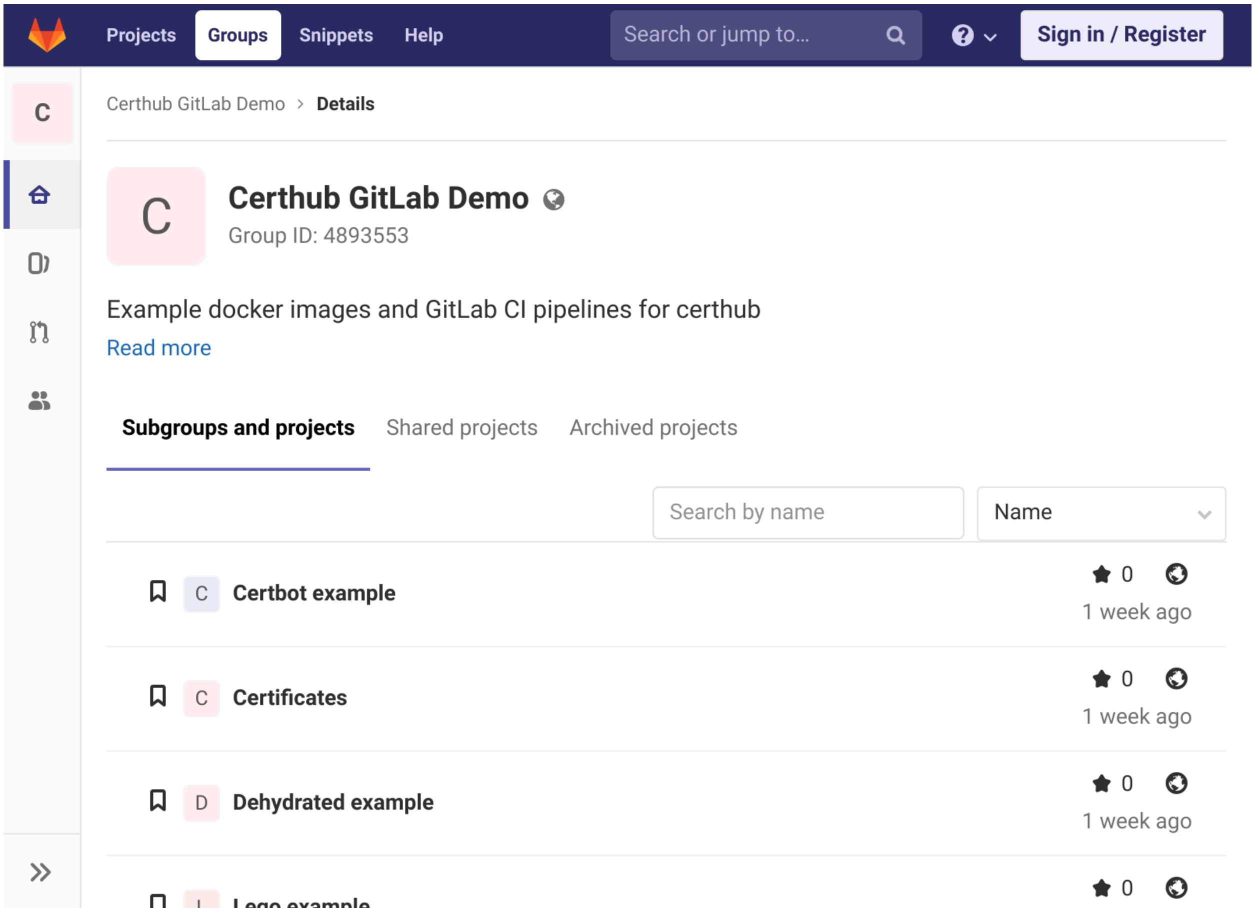
Certhub provides official docker images which can be used as part of a CI pipeline to generate and renew certificates. The images are designed to be fully configurable via environment variables.

Hint: A working demo setup can be found on gitlab.com/certhub-gitlab-demo

Caution: Do not use public/shared CI runners when generating production certificates.

5.1 Big Picture

A typical setup consists of two repositories. One for the CI pipeline (i.e., the Certhub Controller) and a second one for the certificates.



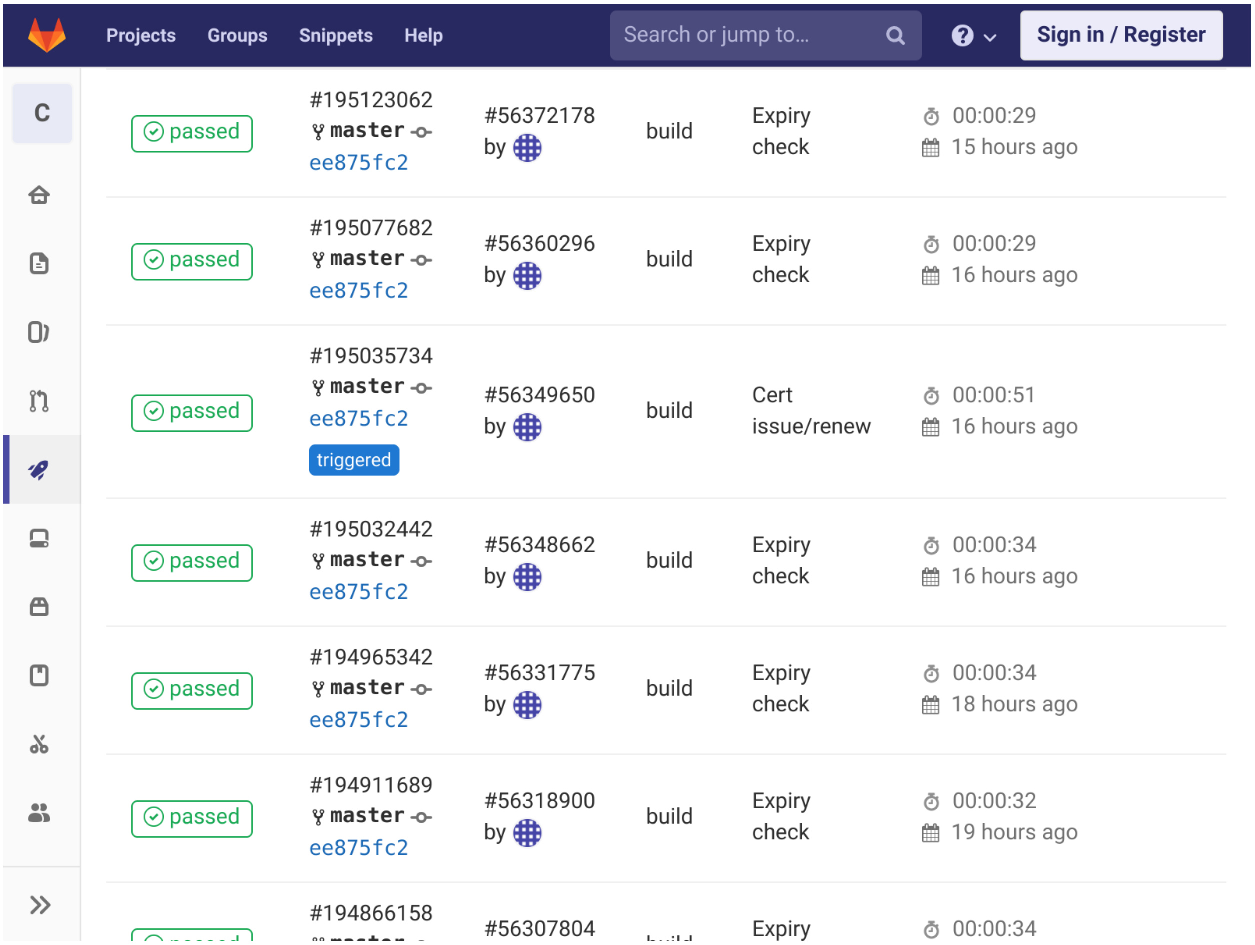
Certhub GitLab Demo

Group ID: 4893553

Example docker images and GitLab CI pipelines for certhub

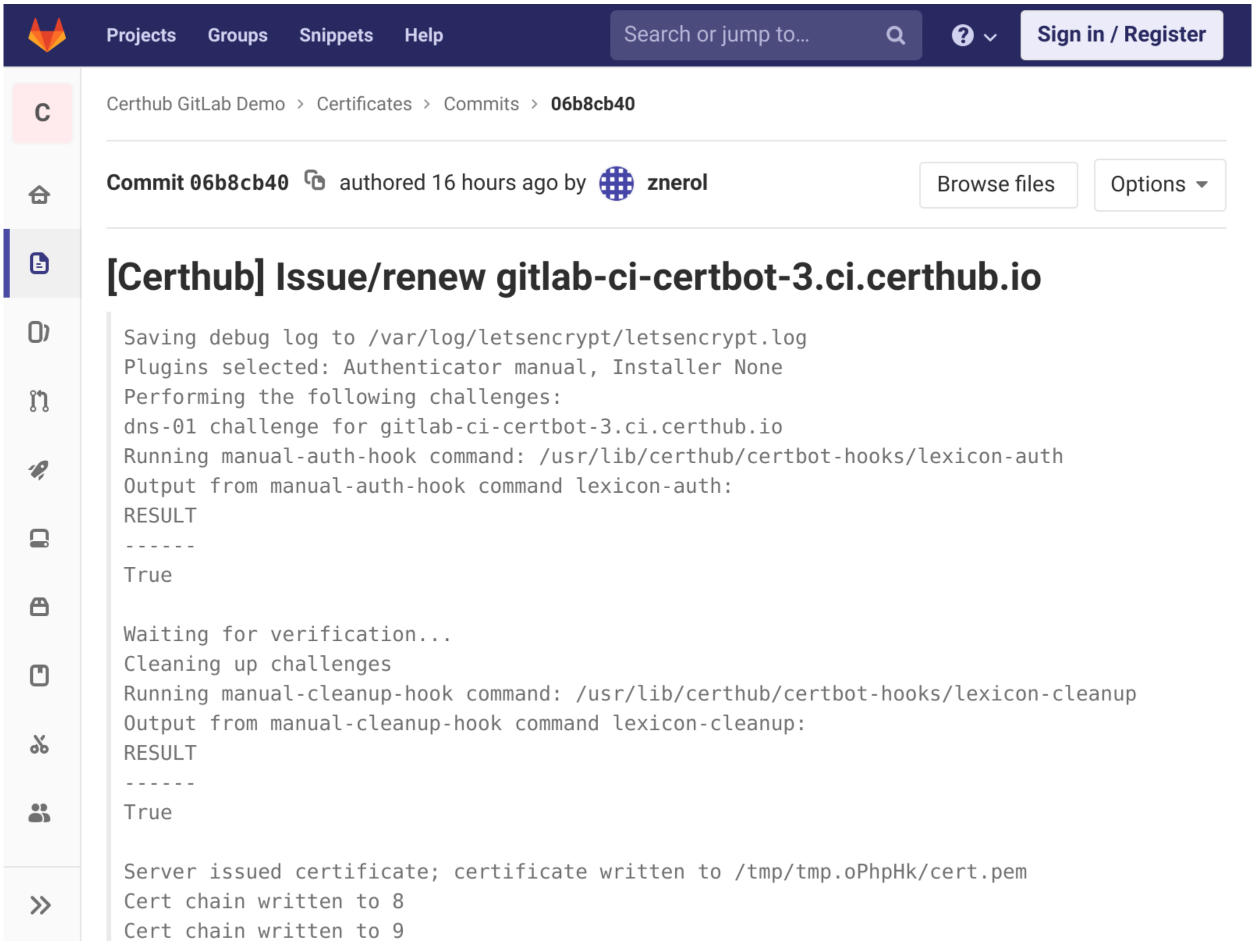
C	Certbot example	0 1 week ago
C	Certificates	0 1 week ago
D	Dehydrated example	0 1 week ago
L	Lego example	0

The CI pipeline itself consists of two jobs: *Cert issue/renew* which only runs when triggered and *Expiry check* which only runs when scheduled. If the *Expiry Check* job detects that a certificate is about to expire, it triggers the *Cert issue/renew* job.



✓ passed	#195123062 master ee875fc2	#56372178 by	build	Expiry check	00:00:29 15 hours ago
✓ passed	#195077682 master ee875fc2	#56360296 by	build	Expiry check	00:00:29 16 hours ago
✓ passed	#195035734 master ee875fc2 triggered	#56349650 by	build	Cert issue/renew	00:00:51 16 hours ago
✓ passed	#195032442 master ee875fc2	#56348662 by	build	Expiry check	00:00:34 16 hours ago
✓ passed	#194965342 master ee875fc2	#56331775 by	build	Expiry check	00:00:34 18 hours ago
✓ passed	#194911689 master ee875fc2	#56318900 by	build	Expiry check	00:00:32 19 hours ago
	#194866158	#56307804		Expiry	00:00:34

If the *Cert isuse/renew* job successfully optains a certificate, it gets pushed to the certificate repository.



Commit 06b8cb40 authored 16 hours ago by znerol

[Certhub] Issue/renew gitlab-ci-certbot-3.ci.certhub.io

```
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator manual, Installer None
Performing the following challenges:
dns-01 challenge for gitlab-ci-certbot-3.ci.certhub.io
Running manual-auth-hook command: /usr/lib/certhub/certbot-hooks/lexicon-auth
Output from manual-auth-hook command lexicon-auth:
RESULT
------
True

Waiting for verification...
Cleaning up challenges
Running manual-cleanup-hook command: /usr/lib/certhub/certbot-hooks/lexicon-cleanup
Output from manual-cleanup-hook command lexicon-cleanup:
RESULT
------
True

Server issued certificate; certificate written to /tmp/tmp.oPhpHk/cert.pem
Cert chain written to 8
Cert chain written to 9
```

5.2 Certificates Repository

In order to setup the certificate repository on gitlab, follow these steps:

1. first create a new project and initialize the repository with a first commit. E.g., add a README file linking to this guide.
2. Generate a new SSH key pair and add the public part as a deploy key to the newly created certificates project. Store the private part of the key in a safe place.

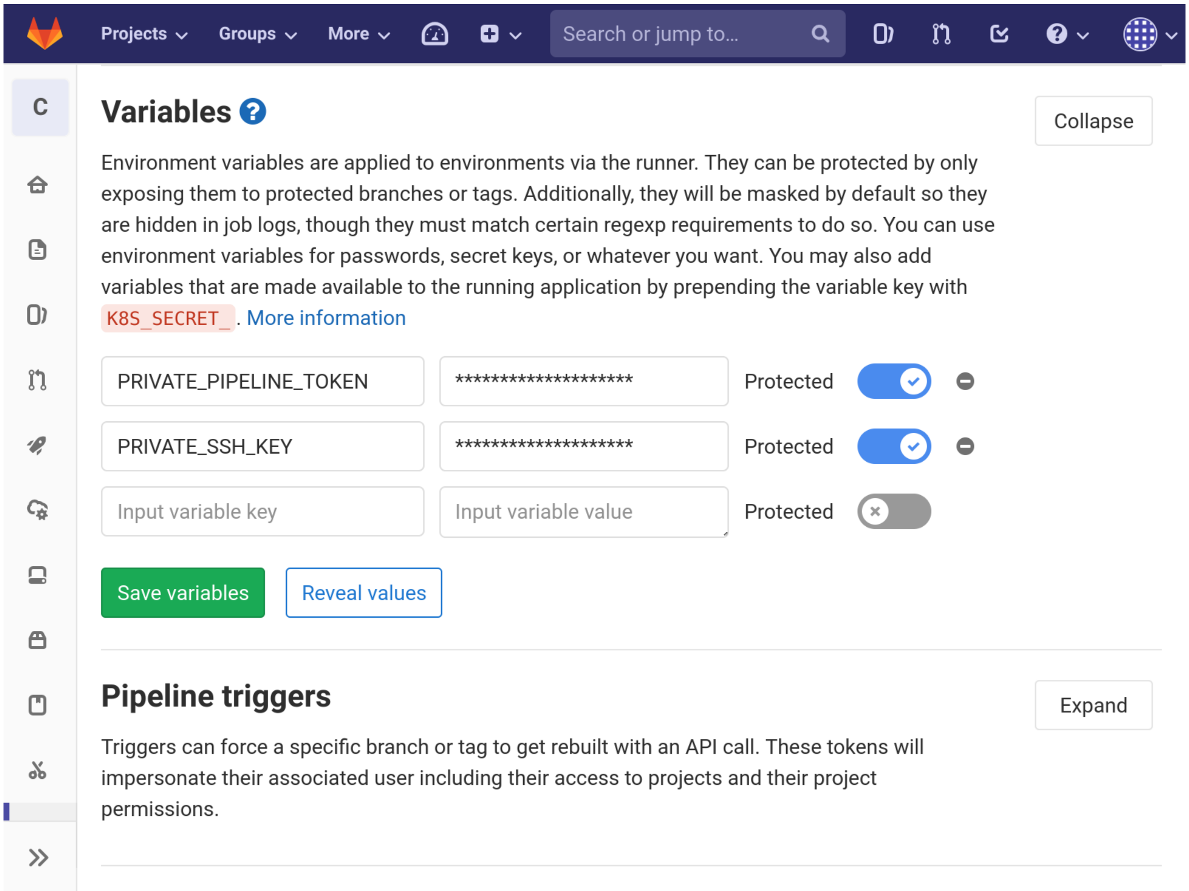
Hint: There are many options to deploy certificates pushed to this repository. Some ideas:

- Setup repository mirroring to push certificates to certhub nodes.
- Trigger other CI Pipelines to rebuild and deploy applications / docker images.
- Setup Webhooks to notify the sysops team.

5.3 CI Pipeline Repository

In order to setup the controller repository and its CI pipeline follow these steps:

1. First create a new project and initialize the repository with a first commit. E.g., add a README file linking to this guide.
2. Add a new trigger and copy the resulting token.
3. Navigate to environment variables section and add the trigger token to a new variable named `PRIVATE_PIPELINE_TOKEN`. Also add the private part of the deploy key as a variable with the name `PRIVATE_SSH_KEY`.



Variables

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they will be masked by default so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. More information

PRIVATE_PIPELINE_TOKEN	********************	Protected
PRIVATE_SSH_KEY	********************	Protected
Input variable key	Input variable value	Protected

Pipeline triggers

Triggers can force a specific branch or tag to get rebuilt with an API call. These tokens will impersonate their associated user including their access to projects and their project permissions.

4. Optional but recommended: Enable the *Protected* option on those variables and setup branch protection for the master branch in order to reduce the risk of leaking credentials.

5.4 CI Pipeline Configuration

Use one of `certhub/certhub:certbot` , `certhub/certhub:dehydrated` or `certhub/certhub:lego` as the base image depending on preference and integration needs. In order to simplify interaction with DNS providers, lexicon is packaged with all images except for the `lego`-one.

The following code example represents the overall architecture of the CI pipeline. The global `variables` section contains connection parameters for the certificates repository, as well as variables defining the repository structure. Note that the `CERT_SLUG` variable will typically be defined manually in the GUI when issuing renewing a certificate for the first time or when setting up a scheduled job. Refer to the variables section of git-gau-docker-entry.1 and the certhub *Man Pages* for detailed information about available environment variables.

The *Cert issue/renew* job is configured to be skipped in a scheduled pipeline run, and it is only executed when `CERT_SLUG` environment variable is set (See only/except docs for more information). The commit message can be customized using variables used by *certhub-message-format*. Note that the git commits are attributed to the user which triggered a pipeline by leveraging predefined variables `GITLAB_USER_NAME` and `GITLAB_USER_EMAIL`.

The *Expiry check* job on the other hand is configured to only run in a scheduled pipeline. The only responsibility is to trigger the CI pipeline whenever the certificate pointed to by `CERT_SLUG` is about to expire. Refer to *certhub-cert-expiry* for information about available configuration options via environment variables.

```
image: certhub/certhub:certbot
#image: certhub/certhub:dehydrated
#image: certhub/certhub:lego

variables:
  # 1. Change Git URL of the certificates repository.
  GAU_REPO: git@gitlab.com:certhub-gitlab-demo/certs.git
  GAU_SSH_PRIVKEY: "${PRIVATE_SSH_KEY}"
  # 2. Use ssh-keyscan to determine the SSH keys of the machine hosting the
  # certificates repository.
  GAU_SSH_KNOWNHOSTS: |
    gitlab.com ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAfuCHKVTjquxvt6CM6tdG4SLp1Btn/
↪nOeHHE5UOzRdf
    gitlab.com ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQCsj2bNKTBSpIYDEGk9KxsGh3mySTRgMtXL583qmBpzeQ+jqCMRgBqB98u3z++J1sKlXH
↪U0tCNyokEi/ueaBMCvbcTHhO7FcwzY92WK4Yt0aGROY5qX2UKSeOvuP4D6TPqKF1onrSzH9bx9XUf2lEdWT/
↪ia1NEKjunUqu1xOB/StKDHMoX4/OKyIzuS0q/
↪T1zOATthvasJFoPrAjkohTyaDUz2LN5JoH839hViyEG82yB+MjcFV5MU3N1l1QL3cVUCh93xSaua1N85qivl+siMkPGbO5xR/
↪En4iEY6K2XPASUEMaieWVNTRCtJ4S8H+9
    gitlab.com ecdsa-sha2-nistp256␣
↪AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFSMqzJeV9rUzU4kWitGjeR4PWSa29SPqJ1fVkhtj3Hw9x
  CERTHUB_CERT_PATH: "{WORKDIR}/${CERT_SLUG}.fullchain.pem"
  CERTHUB_CSR_PATH: "${CERT_SLUG}.csr.pem"
  CERTHUB_CERT_EXPIRY_TTL: 2592000

Cert issue/renew:
  stage: build

  only:
    variables:
      - $CERT_SLUG

  except:
    refs:
      - schedules

  variables:
    CERTHUB_MESSAGE_SUBJECT_ACTION: "Issue/renew ${CERT_SLUG}"

    # 3. Configuration for acme client goes here
    #    [...]


  before_script:
    - git config user.name "${GITLAB_USER_NAME}"
    - git config user.email "${GITLAB_USER_EMAIL}"

  script:
    - >
      git gau-ac
      git gau-xargs -I{WORKDIR}
```

(continues on next page)

(continued from previous page)

```
      certhub-message-format "${CERTHUB_CERT_PATH}" x509
      # 4. Invocation of acme client goes here
      #    [...]

Expiry check:
  stage: build

  only:
    variables:
      - $CERT_SLUG
    refs:
      - schedules

  variables:
    PIPELINE_TOKEN: "${PRIVATE_PIPELINE_TOKEN}"

  script:
    - >
      git gau-xargs -I{WORKDIR}
      certhub-cert-expiry "${CERTHUB_CERT_PATH}" "${CERTHUB_CERT_EXPIRY_TTL}"
      curl -X POST -F "token=${PIPELINE_TOKEN}" -F "ref=${CI_COMMIT_REF_NAME}" -F
↪"variables[CERT_SLUG]=${CERT_SLUG}" "${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/
↪trigger/pipeline"
```

5.5 CI Pipeline Certbot

This section needs work. Please refer to the certbot example on gitlab.com and *certhub-docker-entry*.

5.6 CI Pipeline Dehydrated

This section needs work. Please refer to the dehydrated example on gitlab.com and *certhub-docker-entry*.

5.7 CI Pipeline Lego

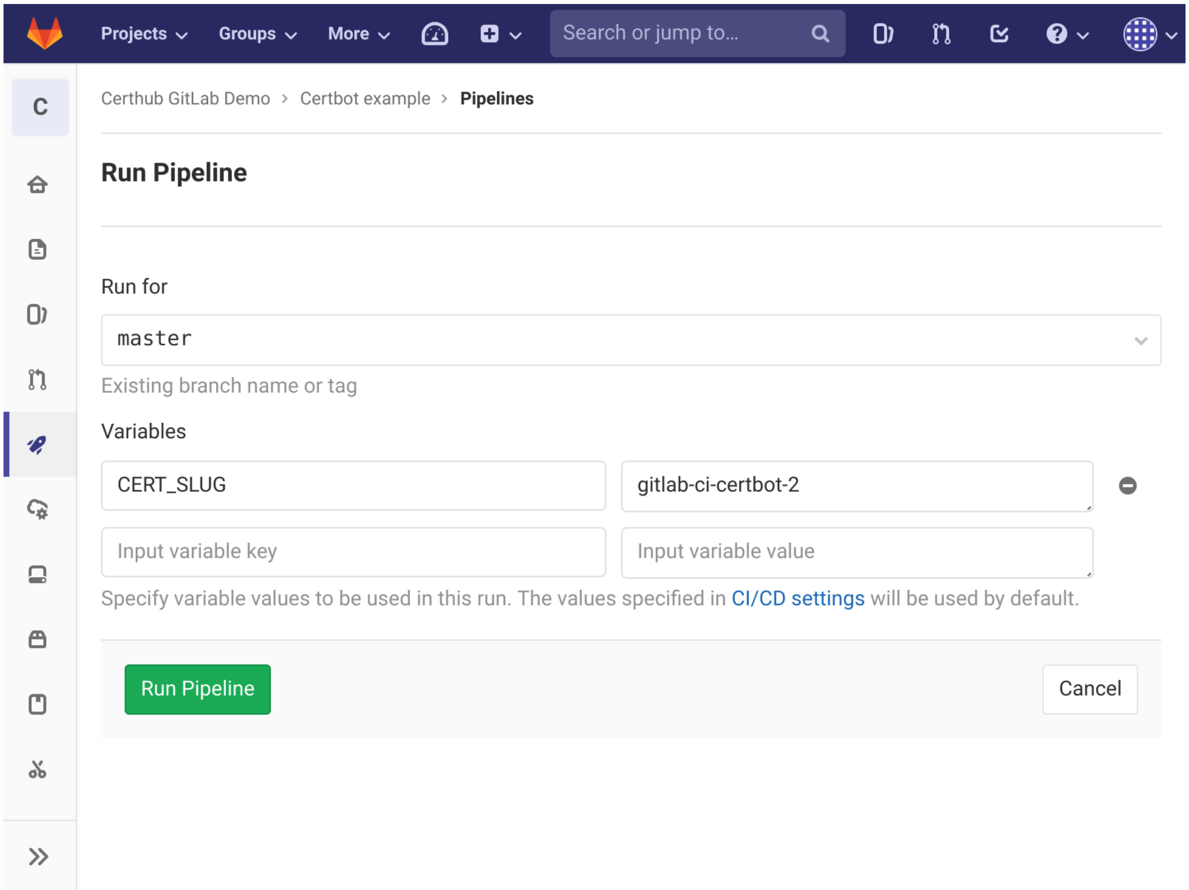
This section needs work. Please refer to the lego example on gitlab.com and *certhub-docker-entry*.

CHAPTER 6

Gitlab CI Usage

In order to setup a new certificate follow these steps:

1. Generate a new private key and a CSR. Store the private key in a safe place and deploy it to the servers / services where the certificate will be used.
2. Add the CSR to the repository (file extension: `.csr.pem`). Note the basename, this will be used as the value of the `CERT_SLUG` variable in subsequent steps.
3. Add acme client specific configuration files to the repository if necessary, (e.g., `$CERT_SLUG.certbot.ini` or `$CERT_SLUG.dehydrated.conf`).
4. Run the CI pipeline once manually, set the variable `CERT_SLUG` in the GUI:



Run Pipeline

Run for

master

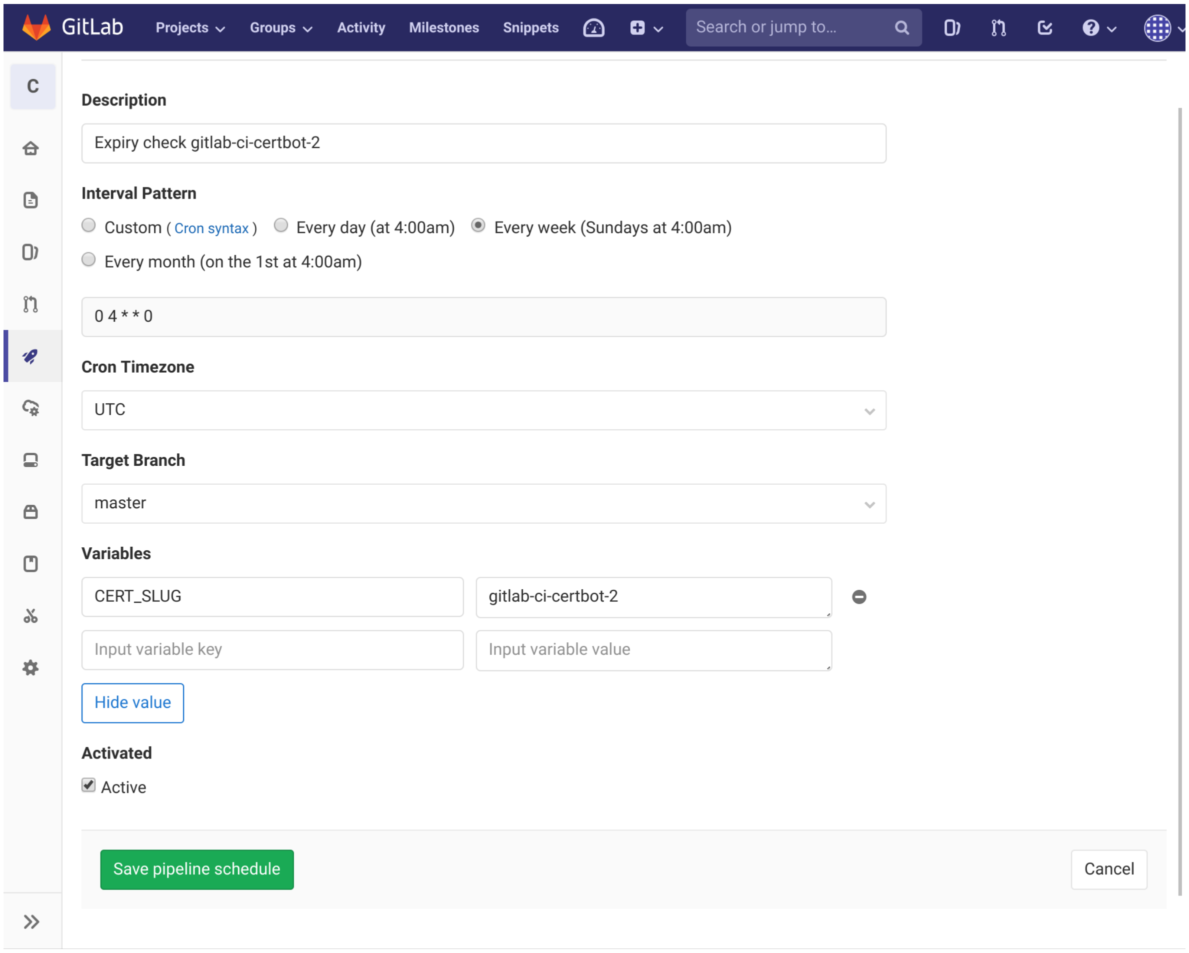
Existing branch name or tag

Variables

CERT_SLUG: gitlab-ci-certbot-2

Specify variable values to be used in this run. The values specified in CI/CD settings will be used by default.

5. Add a schedule if the pipeline was successfull.



Description

Expiry check gitlab-ci-certbot-2

Interval Pattern

○ Custom (Cron syntax) ○ Every day (at 4:00am) ● Every week (Sundays at 4:00am)
○ Every month (on the 1st at 4:00am)

0 4 * * 0

Cron Timezone

UTC

Target Branch

master

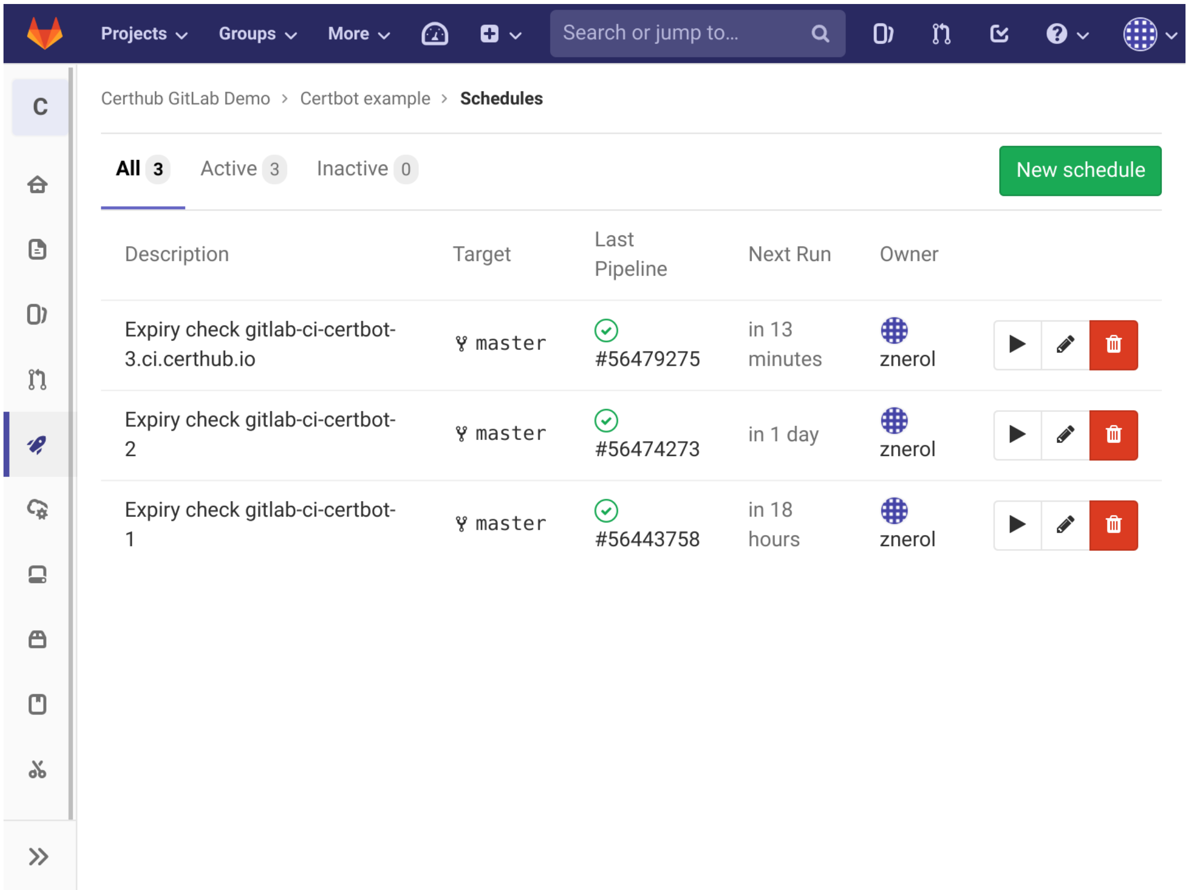
Variables

CERT_SLUG gitlab-ci-certbot-2

Activated

☑ Active

6. Expiry checks can also be triggered manually from the schedule overview page.



Certhub GitLab Demo › Certbot example › Schedules

Description	Target	Last Pipeline	Next Run	Owner
Expiry check gitlab-ci-certbot-3.ci.certhub.io	`master`	✓ #56479275	in 13 minutes	zneroI
Expiry check gitlab-ci-certbot-2	`master`	✓ #56474273	in 1 day	zneroI
Expiry check gitlab-ci-certbot-1	`master`	✓ #56443758	in 18 hours	zneroI

CHAPTER 7

Man Pages

7.1 certhub-certbot-run

7.1.1 Synopsis

certhub-certbot-run <*output-cert-file*> <*input-csr-file*> <*certbot*> [*certbot-certonly-args* . . .]

7.1.2 Description

Runs the given **`certbot`** binary with CSR read from `<input-csr-file>`. Writes the resulting certificate to the `<output-cert-file>` as well.

7.1.3 Examples

Run **`certbot certonly`** with CSR from the configuration directory. Resulting fullchain certificate is committed to the repository.

```
git gau-exec /var/lib/certhub/certs.git \
git gau-ac \
git gau-xargs -I{} \
certhub-message-format {}/example.com.fullchain.pem x509 \
certhub-certbot-run {}/example.com.fullchain.pem /etc/certhub/example.com.csr.pem \
certbot --config /etc/certhub/example.com.certbot.ini
```

7.1.4 See Also

`certbot(1)`, *`certhub-message-format(1)`*

7.2 certhub-dehydrated-run

7.2.1 Synopsis

certhub-dehydrated-run <*output-cert-file*> <*input-csr-file*> <*dehydrated*> [*dehydrated-args* …]

7.2.2 Description

Runs the given **`dehydrated`** binary with CSR read from `<input-csr-file>` Writes the resulting certificate to the `<output-cert-file>`.

7.2.3 Examples

Run **`dehydrated --signcsr`** with CSR from the configuration directory. Resulting fullchain certificate is committed to the repository.

```
git gau-exec /var/lib/certhub/certs.git \
git gau-ac \
git gau-xargs -I{} \
certhub-message-format {}/example.com.fullchain.pem x509 \
certhub-dehydrated-run {}/example.com.fullchain.pem /etc/certhub/example.com.csr.pem \
dehydrated --config /etc/certhub/example.com.dehydrated.conf
```

7.2.4 See Also

`dehydrated(1)`, *`certhub-message-format(1)`*

7.3 certhub-lego-run

7.3.1 Synopsis

certhub-lego-run <*output-cert-file*> <*input-csr-file*> <*lego-directory*> <*lego*> [*lego-run-args* …]

certhub-lego-run-preferred-chain <*preferred-chain*> <*output-cert-file*> <*input-csr-file*> <*lego-directory*> <*lego*> [*lego-run-args* …]

7.3.2 Description

Runs the given **`lego`** binary with CSR read from `<input-csr-file>`. Writes the resulting certificate to the `<output-cert-file>`.

Note, `<lego-directory>` must point to the directory where lego stores account data and certificates (usually `$HOME/.lego`).

In order to specify the preferred-chain, use the `certhub-lego-run-preferred-chain` binary and specify the CN of the preferred root certificate as the first argument.

7.3.3 Examples

Run `lego run` with CSR from configuration directory. Resulting fullchain certificate is committed to the repository.

```
git gau-exec /var/lib/certhub/certs.git \
git gau-ac \
git gau-xargs -I{} \
certhub-message-format {}/example.com.fullchain.pem x509 \
certhub-lego-run {}/example.com.fullchain.pem /etc/certhub/example.com.csr.pem /var/
↪lib/certhub/private/lego \
lego --accept-tos --email hello@example.com
```

Run `lego run` with CSR from configuration directory and request a certificate with the alternate/short Let's Encrypt certificate chain. Resulting fullchain certificate is committed to the repository.

```
git gau-exec /var/lib/certhub/certs.git \
git gau-ac \
git gau-xargs -I{} \
certhub-message-format {}/example.com.fullchain.pem x509 \
certhub-lego-run-preferred-chain "ISRG Root X1" {}/example.com.fullchain.pem /etc/
↪certhub/example.com.csr.pem /var/lib/certhub/private/lego \
lego --accept-tos --email hello@example.com
```

7.3.4 See Also

certhub-message-format (1)

7.4 certhub-cert-expiry

7.4.1 Synopsis

certhub-cert-expiry <*input-cert-file*> <*seconds*> <*command*> [*args* . . .]

7.4.2 Description

If the given certificate is about to expire in the given amount of seconds, run the command.

Common values for the ttl parameter:

86400 Twenty four hours.

604800 7 days.

2592000 30 days.

7.4.3 Examples

Run `certhub-cert-expiry` with certificate read from the repository. Format a message containing information about the certificate and write it to the status file if its expiration date is within 30 days.

```
git gau-exec /var/lib/certhub/certs.git \
git gau-xargs -I{} \
certhub-status-file /var/lib/certhub/status/example.com.expiry.status
certhub-cert-expiry "{}/example.com.fullchain.pem" 2592000 \
certhub-message-format "{}/example.com.fullchain.pem" x509 \
echo "Certificate will expire within 30 days"
```

7.4.4 See Also

`certhub-status-file(1)`

7.5 certhub-message-format

7.5.1 Synopsis

certhub-message-format <*input-pem-file*> [*x509*|*req*] <*command*> [*args* . . .]

7.5.2 Description

Runs the specified `<command>` and capture its standard output and standard error. Formats the output in a way which suites **`git-commit`** / **`git-gau-ac`**. Also attaches CSR or certificate details if the specified `<input-pem-file>` exists.

Use this command in combination with **`certhub-certbot-run`** and **`git-gau-exec`** / **`git-gau-ac`** when adding / renewing certificates in an automated way.

7.5.3 Environment

`CERTHUB_MESSAGE_SUBJECT`
First line of the message. By default this is generated automatically.

`CERTHUB_MESSAGE_SUBJECT_PREFIX`
Message prefix when automated subject generation is enabled. Defaults to `[Certhub]`.

`CERTHUB_MESSAGE_SUBJECT_ACTION`
Message action name when automated subject generation is enabled. Defaults to basename of executed command.

`CERTHUB_MESSAGE_CSR_TEXTOPTS`
Output options as understood by **`openssl req`**. Defaults to: `--noout -text -reqopt no_pubkey,no_sigdump`

`CERTHUB_MESSAGE_CERT_TEXTOPTS`
Output options as understood by **`openssl x509`**. Defaults to: `--noout -text -certopt no_pubkey,no_sigdump,no_extensions -sha256 -fingerprint`

7.5.4 See Also

`certhub-certbot-run(1)`

7.6 certhub-send-file

7.6.1 Synopsis

certhub-send-file <*input-file*> <*command*> [*args* …]

7.6.2 Description

Pipes the given file to standard input of the specified `<command>`.

7.7 certhub-status-file

7.7.1 Synopsis

certhub-status-file <*output-status-file*> <*command*> [*args* …]

7.7.2 Description

Runs the specified `<command>` and capture its standard output. Writes the output to the specified status file. Removes the status file if command doesn't output anything.

Use this command in combination with **`certhub-cert-expiry`** in order to flag certificates which are about to expire.

7.7.3 See Also

`certhub-cert-expiry(1)`

7.8 certhub-certbot-run@.service

7.8.1 Synopsis

certhub-certbot-run@.service

certhub-certbot-run@.path

7.8.2 Description

A service which runs **`certhub-certbot-run`** with a CSR read from the config directory. The resulting fullchain certificate is committed to the repository. A commit message is generated automatically.

A path unit which runs the service unit if the expiry status file managed by **`certhub-cert-expiry@.service`** exists or if the CSR file changed.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the resulting certificate file.

7.8.3 Environment

`CERTHUB_REPO`
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

`CERTHUB_CERT_PATH`
Path to the certificate file inside the repository. Defaults to: `{WORKDIR}/%i.fullchain.pem`

`CERTHUB_CSR_PATH`
Path to the CSR file. Defaults to: `/etc/certhub/%i.csr.pem`

`CERTHUB_CERTBOT_ARGS`
Additional Arguments for **`certbot certonly`** run. Defaults to: `--non-interactive`

`CERTHUB_CERTBOT_CONFIG`
Path to a certbot configuration file. Defaults to: `/etc/certhub/%i.certbot.ini`

7.8.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-certbot-run.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-certbot-run.env`
Optional per-instance and per-service environment file.

7.8.5 See Also

`certhub-cert-expiry@.service`, *`certhub-certbot-run(1)`*, *`certhub-message-format(1)`*

7.9 certhub-dehydrated-run@.service

7.9.1 Synopsis

certhub-dehydrated-run@.service

certhub-dehydrated-run@.path

7.9.2 Description

A service which runs **`certhub-dehydrated-run`** with a CSR read from the config directory. The resulting fullchain certificate is committed to the repository. A commit message is generated automatically.

A path unit which runs the service unit if the expiry status file managed by **`certhub-cert-expiry@.service`** exists or if the CSR file changed.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the resulting certificate file.

7.9.3 Environment

`CERTHUB_REPO`
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

`CERTHUB_CERT_PATH`
Path to the certificate file inside the repository. Defaults to: `{WORKDIR}/%i.fullchain.pem`

`CERTHUB_CSR_PATH`
Path to the CSR file. Defaults to: `/etc/certhub/%i.csr.pem`

`CERTHUB_DEHYDRATED_ARGS`
Additional Arguments for **`dehydrated --signcsr`** run. Empty by default.

`CERTHUB_DEHYDRATED_CONFIG`
Path to a dehydrated configuration file. Defaults to: `/etc/certhub/%i.dehydrated.conf`

7.9.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-dehydrated-run.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-dehydrated-run.env`
Optional per-instance and per-service environment file.

7.9.5 See Also

`certhub-cert-expiry@.service`, *`certhub-dehydrated-run(1)`*, *`certhub-message-format(1)`*

7.10 certhub-lego-run@.service

7.10.1 Synopsis

certhub-lego-run@.service

certhub-lego-run@.path

7.10.2 Description

A service which runs **`certhub-lego-run`** with a CSR read from the config directory. The resulting fullchain certificate is committed to the repository. A commit message is generated automatically.

A path unit which runs the service unit if the expiry status file managed by **`certhub-cert-expiry@.service`** exists or if the CSR file changed.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the resulting certificate file.

7.10.3 Environment

CERTHUB_REPO
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

CERTHUB_CERT_PATH
Path to the certificate file inside the repository. Defaults to: `{WORKDIR}/%i.fullchain.pem`

CERTHUB_CSR_PATH
Path to the CSR file. Defaults to: `/etc/certhub/%i.csr.pem`

CERTHUB_LEGO_ARGS
Additional Arguments for **lego --csr** run. Empty by default.

CERTHUB_LEGO_PREFERRED_CHAIN
Set the preferred certificate chain. If the CA offers multiple certificate chains, prefer the chain whose topmost certificate was issued from this Subject Common Name. If no match, the default offered chain will be used. Empty by default.

Specify `CERTHUB_LEGO_PREFERRED_CHAIN=ISRG Root X1` in one of the envfiles listed in the next section to use the alternate/short Let's Encrypt chain.

CERTHUB_LEGO_CHALLENGE_ARGS
Use this environment variable to select a challenge method. Empty by default. Lego will fall back to HTTP-01 challenge if this variable is not set.

CERTHUB_LEGO_DIR
The path to the directory where lego stores accound data and issued certificates. Defaults to: `var/lib/certhub/private/lego`

7.10.4 Files

/etc/certhub/env
Optional environment file shared by all instances and certhub services.

/etc/certhub/%i.env
Optional per-instance environment file shared by all certhub services.

/etc/certhub/certhub-lego-run.env
Optional per-service environment file shared by all certhub service instances.

/etc/certhub/%i.certhub-lego-run.env
Optional per-instance and per-service environment file.

7.10.5 See Also

`certhub-cert-expiry@.service`, *`certhub-lego-run(1)`*, *`certhub-message-format(1)`*

7.11 certhub-cert-expiry@.service

7.11.1 Synopsis

certhub-cert-expiry@.service

certhub-cert-expiry@.path

certhub-cert-expiry@.timer

7.11.2 Description

A service which checks validity of a certificate read from the repository. Formats a message and writes it to a status file if the respective certificate is about to expire.

A path unit which runs the service unit whenever the master branch of the local certhub repository is updated.

A timer unit which runs the service twice daily.

The instance name (systemd instance string specifier `%i`) is used as the basename of the certificate file and the resulting status message.

7.11.3 Environment

`CERTHUB_REPO`
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

`CERTHUB_CERT_PATH`
Path to the certificate file inside the repository. Defaults to: `{WORKDIR}/%i.fullchain.pem`

`CERTHUB_CERT_EXPIRY_TTL`
See manpage:*certhub-cert-expiry(1)*, defaults to 30 days in seconds, i.e. `2592000`

`CERTHUB_CERT_EXPIRY_MESSAGE`
Message written to the status file if certificate is about to expire. Defaults to `Certificate will expire within 30 days`

`CERTHUB_CERT_EXPIRY_STATUSFILE`
Location of status file written if a certificate is about to expire. Defaults to: `/var/lib/certhub/status/%i.expiry.status`

7.11.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-cert-expiry.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-cert-expiry.env`
Optional per-instance and per-service environment file.

7.11.5 See Also

`certhub-cert-expiry(1)`, *`certhub-format-message(1)`*, *`certhub-status-file(1)`*

7.12 certhub-cert-export@.service

7.12.1 Synopsis

certhub-cert-export@.service

certhub-cert-export@.path

7.12.2 Description

A service which copies a certificate from the repository to the local filesystem.

A path unit which runs the service unit whenever the master branch of the local certhub repository is updated.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the resulting certificate file.

7.12.3 Environment

`CERTHUB_REPO`
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

`CERTHUB_CERT_EXPORT_SRC`
File / directory inside the repository which should be exported. Defaults to: `{WORKDIR}/%i.fullchain.pem`

`CERTHUB_CERT_EXPORT_DEST`
File / directory where the certificate should be placed. Defaults to: `/var/lib/certhub/certs/%i.fullchain.pem`

`CERTHUB_CERT_EXPORT_RSYNC_ARGS`
Arguments for rsync. Defaults to: `--checksum --delete --devices --links --perms --recursive --specials --verbose`

7.12.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-cert-export.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-cert-export.env`
Optional per-instance and per-service environment file.

7.12.5 See Also

`rsync(1)`

7.13 certhub-cert-reload@.service

7.13.1 Synopsis

certhub-cert-reload@.service

certhub-cert-reload@.path

7.13.2 Description

A service which reloads specified tls servers.

A path unit which runs the service unit whenever the exported certificate has changed on the filesystem.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the certificate file.

7.13.3 Environment

`CERTHUB_CERT_RELOAD_CONFIG`
Path to a file containing the services to reload. Defaults to: `/etc/certhub/%i.services-reload.txt`

`CERTHUB_CERT_RELOAD_COMMAND`
A systemctl subcommand to execute when a service needs to be reloaded. Useful values include `reload`, `restart`, `try-restart`, `reload-or-restart`, `try-reload-or-restart`. Defaults to `reload`.

7.13.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-cert-reload.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-cert-reload.env`
Optional per-instance and per-service environment file.

7.13.5 See Also

`certhub-cert-export@.service(8)`

7.14 certhub-cert-send@.service

7.14.1 Synopsis

certhub-cert-send@.service

certhub-cert-send@.path

7.14.2 Description

A service which a sends certificate to predefined destinations. The specified command line is executed once for each destination with the certificate piped to stdin.

A path unit which runs the service unit whenever the exported certificate has changed on the filesystem.

The instance name (systemd instance string specifier `%i`) is used as the basename of the configuration and the certificate file.

7.14.3 Environment

`CERTHUB_CERT_SEND_SRC`
Path to the certificate file to be sent. Defaults to: `/var/lib/certhub/certs/%i.fullchain.pem`

`CERTHUB_CERT_SEND_CONFIG`
Path to a file containing destinations for the certificate send service. `/etc/certhub/%i.destinations-send.txt`

`CERTHUB_CERT_SEND_COMMAND`
Command to execute for each predefined destination. Use `%i` to reference the instance name. The command is run once for each line in the `%i.destinations-send.txt` config file, the line can be referenced with the `{DESTINATION}}` placeholder.

Defaults to: `mail -s '[Certhub] Issue/renew %i' {DESTINATION}`

7.14.4 Examples

Configuration file (placed in `/etc/certhub/%i.destinations-send.txt`) containing one destination to send the certificate to `root@localhost` whenever a new one has been exported.

```
root@localhost
```

7.14.5 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-cert-send.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-cert-send.env`
Optional per-instance and per-service environment file.

7.14.6 See Also

`certhub-cert-export@.service(8)`

7.15 certhub-repo-push@.service

7.15.1 Synopsis

certhub-repo-push@.service

certhub-repo-push@.path

7.15.2 Description

A service which pushes the certhub repository to another host.

A path unit which runs the service unit whenever the master branch of the local certhub repository is updated.

The unescaped instance name (systemd unescaped instance string specifier `%I`) is used as the URL for the remote repository. Note that the instance name likely needs to be escaped using **`systemd-escape --template`**.

7.15.3 Environment

`CERTHUB_REPO`
URL of the repository where certificates are stored. Defaults to: `/var/lib/certhub/certs.git`

`CERTHUB_REPO_PUSH_REMOTE`
URL of the remote repository. Defaults to: `CERTHUB_REPO_PUSH_REMOTE=%I`

`CERTHUB_REPO_PUSH_ARGS`
Arguments to the git push command. Defaults to: `--mirror`

`CERTHUB_REPO_PUSH_REFSPEC`
Refspec for the git push command. Empty by default.

7.15.4 Files

`/etc/certhub/env`
Optional environment file shared by all instances and certhub services.

`/etc/certhub/%i.env`
Optional per-instance environment file shared by all certhub services.

`/etc/certhub/certhub-repo-push.env`
Optional per-service environment file shared by all certhub service instances.

`/etc/certhub/%i.certhub-repo-push.env`
Optional per-instance and per-service environment file.

7.15.5 See Also

`git-push(1)`

7.16 certhub-docker-entry

7.16.1 Synopsis

/usr/local/lib/git-gau/docker-entry.d/60-acme-dns-registration

/usr/local/lib/git-gau/docker-entry.d/60-certbot-account

/usr/local/lib/git-gau/docker-entry.d/60-dehydrated-account

/usr/local/lib/git-gau/docker-entry.d/60-lego-account

7.16.2 Description

A collection of docker entrypoint scripts called by *git-gau* **`docker-entry`** via **`run-parts`**. Useful to setup preexisting ACME accounts from data passed into a container by environment variables.

Refer to *`git-gau-docker-entry(8)`* for more information on the entrypoint scripts shipping with *git-gau*. Note that for common use cases *GAU_REPO* should point to the certhub certificate repository.

7.16.3 Environment (acme-dns)

It is recommended to specify *`CERTHUB_ACME_DNS_REGISTRATION`* for a production setup when using joohoi/acme-dns.

`CERTHUB_ACME_DNS_REGISTRATION`
Contents of the JSON registration file as generated by **`goacmedns-register`** which is part of cpu/goacmedns. Note that more than one account can be registered/represented in a single JSON data structure.

`CERTHUB_ACME_DNS_REGISTRATION_FILE`
Full path to registration json file. Defaults to `${HOME}/acme-dns-registration.json`.

7.16.4 Environment (Certbot)

It is recommended to specify *`CERTHUB_CERTBOT_ACCOUNT_ID`*, *`CERTHUB_CERTBOT_ACCOUNT_KEY`*, *`CERTHUB_CERTBOT_ACCOUNT_REGR`* and *`CERTHUB_CERTBOT_ACCOUNT_META`* for a production setup. The remaining variables can be ignored in most situations.

`CERTHUB_CERTBOT_ACCOUNT_KEY`
ACME account private key in JSON format used by certbot. If this variable is non-empty, its contents will be written to *private_key.json* in the respective accounts directory. Note that either *`CERTHUB_CERTBOT_ACCOUNT_ID`* or *`CERTHUB_CERTBOT_ACCOUNT_DIR`* is required if this variable is set.

`CERTHUB_CERTBOT_ACCOUNT_REGR`
ACME account registration information in JSON format used by certbot. If this variable is non-empty, its contents will be written to *regr.json* in the respective accounts directory. Note that either *`CERTHUB_CERTBOT_ACCOUNT_ID`* or *`CERTHUB_CERTBOT_ACCOUNT_DIR`* is required if this variable is set.

`CERTHUB_CERTBOT_ACCOUNT_META`
ACME account meta information in JSON format used by certbot. If this variable is non-empty, its contents will be written to *meta.json* in the respective accounts directory. Note that either

CERTHUB_CERTBOT_ACCOUNT_ID or *CERTHUB_CERTBOT_ACCOUNT_DIR* is required if this variable is set.

CERTHUB_CERTBOT_ACCOUNT_ID
ACME account id as used by certbot to identify the account in the form of a 32 character long hex string. This is equivalent to the last component of an account directory path.

CERTHUB_CERTBOT_ACCOUNT_SERVER
ACME endpoint URL for the given account. Defaults to: *https://acme-v02.api.letsencrypt.org/directory*

CERTHUB_CERTBOT_CONFIG_DIR
Base directory for certbot configuration. Defaults to: */etc/letsencrypt*.

CERTHUB_CERTBOT_ACCOUNT_DIR
Full path to an accounts directory. Defaults to a value computed from *CERTHUB_CERTBOT_CONFIG_DIR*, *CERTHUB_CERTBOT_ACCOUNT_SERVER* and *CERTHUB_CERTBOT_ACCOUNT_ID*.

7.16.5 Environment (Dehydrated)

It is recommended to specify *CERTHUB_DEHYDRATED_ACCOUNT_KEY*, *CERTHUB_DEHYDRATED_ACCOUNT_REGR* and *CERTHUB_DEHYDRATED_ACCOUNT_ID* for a production setup. The remaining variables can be ignored in most situations.

CERTHUB_DEHYDRATED_ACCOUNT_KEY
ACME account private key in PEM format used by dehydrated. If this variable is non-empty, its contents will be written to *account_key.pem* in the respective accounts directory.

CERTHUB_DEHYDRATED_ACCOUNT_REGR
ACME account registration information in JSON format used by dehydrated. If this variable is non-empty, its contents will be written to *registration_info.json* in the respective accounts directory. set.

CERTHUB_DEHYDRATED_ACCOUNT_ID
ACME account id information in JSON format used by dehydrated. If this variable is non-empty, its contents will be written to *account_id.json* in the respective accounts directory.

CERTHUB_DEHYDRATED_ACCOUNT_SERVER
ACME endpoint URL for the given account. Defaults to: *https://acme-v02.api.letsencrypt.org/directory*

CERTHUB_DEHYDRATED_CONFIG_DIR
Base directory for dehydrated configuration. Defaults to: */etc/dehydrated*.

CERTHUB_DEHYDRATED_ACCOUNT_DIR
Full path to an accounts directory. Defaults to a value computed from *CERTHUB_DEHYDRATED_CONFIG_DIR* and *CERTHUB_DEHYDRATED_ACCOUNT_SERVER*.

7.16.6 Environment (Lego)

It is recommended to specify *CERTHUB_LEGO_ACCOUNT_EMAIL* *CERTHUB_LEGO_ACCOUNT_KEY* and *CERTHUB_LEGO_ACCOUNT_CONF* for a production setup. The remaining variables can be ignored in most situations.

CERTHUB_LEGO_ACCOUNT_KEY
ACME account private key in PEM format used by lego. If this variable is non-empty, its contents will be written to *${CERTHUB_LEGO_ACCOUNT_EMAIL}.key* in the respective accounts directory. Note that either *CERTHUB_LEGO_ACCOUNT_EMAIL* or *CERTHUB_LEGO_ACCOUNT_KEY_DIR*/*CERTHUB_LEGO_ACCOUNT_KEY_FILE* are required if this variable is set.

CERTHUB_LEGO_ACCOUNT_CONF
ACME account registration information in JSON format used by lego. If this variable is non-empty, its contents will be written to *account.json* in the respective accounts directory. Note that either *CERTHUB_LEGO_ACCOUNT_EMAIL* or *CERTHUB_LEGO_ACCOUNT_DIR*/*CERTHUB_LEGO_ACCOUNT_CONF_FILE* are required if this variable is set.

CERTHUB_LEGO_ACCOUNT_EMAIL
ACME account email as used by lego to identify the account.

CERTHUB_LEGO_ACCOUNT_SERVER
ACME endpoint URL for the given account. Defaults to: *https://acme-v02.api.letsencrypt.org/directory*

CERTHUB_LEGO_DIR
Base directory for lego configuration. Defaults to: *${HOME}/.lego*.

CERTHUB_LEGO_ACCOUNT_DIR
Full path to an accounts directory. Defaults to a value computed from *CERTHUB_LEGO_DIR*, *CERTHUB_LEGO_ACCOUNT_SERVER* and *CERTHUB_LEGO_ACCOUNT_EMAIL*.

CERTHUB_LEGO_ACCOUNT_CONF_FILE
Full path to an accounts config file. Defaults to a value computed from *CERTHUB_LEGO_DIR*, *CERTHUB_LEGO_ACCOUNT_SERVER* and *CERTHUB_LEGO_ACCOUNT_EMAIL*.

CERTHUB_LEGO_ACCOUNT_KEY_DIR
Full path to an accounts key directory. Defaults to a value computed from *CERTHUB_LEGO_DIR*, *CERTHUB_LEGO_ACCOUNT_SERVER* and *CERTHUB_LEGO_ACCOUNT_EMAIL*.

CERTHUB_LEGO_ACCOUNT_KEY_FILE
Full path to an accounts key file. Defaults to a value computed from *CERTHUB_LEGO_DIR*, *CERTHUB_LEGO_ACCOUNT_SERVER* and *CERTHUB_LEGO_ACCOUNT_EMAIL*.

7.16.7 See Also

git-gau-docker-entry(8),

7.17 certhub-hook-lexicon-auth

7.17.1 Synopsis

/usr/local/lib/certhub/certbot-hooks/hook-lexicon-auth

/usr/local/lib/certhub/certbot-hooks/hook-lexicon-cleanup

/usr/local/lib/certhub/dehydrated-hooks/hook-lexicon-auth

7.17.2 Description

A hook script for **certbot** and **dehydrated** respectively capable of deploying DNS-01 challenge tokens via **lexicon**.

7.17.3 Environment

`CERTHUB_LEXICON_PROVIDER`
Specify the domain provider which hosts the zone to be used in the DNS challenge.

`CERTHUB_LEXICON_CREATE_EXIT_DELAY`
Optional delay in seconds after record creation (defaults to 5):

`CERTHUB_LEXICON_GLOBAL_ARGS`
Optional additional global lexicon arguments: see *`lexicon(1)`* for more information.

`CERTHUB_LEXICON_PROVIDER_ARGS`
Optional additional lexicon provider arguments (e.g. logging): see *`lexicon(1)`* for more information.

`CERTHUB_LEXICON_DOMAIN`
Domain name passed to lexicon to use for the challenge. Defaults to ${domain-to-be-validated}. Customizing this setting makes sense, e.g. when using `CNAME` records to redirect _acme-challenge names from the real domain to a separate zone purpose built for challange validation.

`CERTHUB_LEXICON_NAME`
Record name to use for the challenge. Defaults to `_acme-challenge.${CERTHUB_LEXICON_DOMAIN}`. Customizing this setting makes sense, e.g. when using `CNAME` records to redirect _acme-challenge names from the real domain to a separate zone purpose built for challange validation.

7.17.4 See Also

`lexicon(1)`,

7.18 certhub-hook-nsupdate-auth

7.18.1 Synopsis

/usr/local/lib/certhub/certbot-hooks/hook-nsupdate-auth

/usr/local/lib/certhub/certbot-hooks/hook-nsupdate-cleanup

/usr/local/lib/certhub/dehydrated-hooks/hook-nsupdate-auth

7.18.2 Description

A hook script for **`certbot`** and **`dehydrated`** respectively capable of deploying DNS-01 challenge tokens via **`nsupdate`**.

7.18.3 Environment

`CERTHUB_NSUPDATE_ARGS`
Arguments passed to nsupdate called from auth/cleanup hooks. Specify the path to the DDNS key used to update a DNS zone. Example: `CERTHUB_NSUPDATE_ARGS=-k /etc/certhub/example.com.nsupdate.key`

`CERTHUB_NSUPDATE_SERVER`
Contact the specified server. By default nsupdate queries SOA records in order to determine the authoritative server. Example: `CERTHUB_NSUPDATE_SERVER=some-ns.example.com`

CERTHUB_NSUPDATE_TTL
TTL for created DNS records. Defaults to 600.

CERTHUB_NSUPDATE_DOMAIN
Domain name to use for the challenge. Uses `_acme-challenge.${domain-to-be-validated}` by default. Customizing this setting makes sense, e.g. when using `CNAME` records to redirect _acme-challenge names from the real domain to a separate zone purpose built for challange validation.

7.18.4 See Also

`nsupdate(1)`,

CHAPTER 8

Best Practice

8.1 Generating TLS Keys and Signing Requests

Use a trustworthy machine with good entropy to generate TLS keys.

8.1.1 RSA Keys

SSL Labs recommends a key size of 2048 bits for most use cases. They discourage usage of keys bigger than 3072 bits. Use the following command to generate RSA keys with `openssl`.

```
# 2048 bit RSA
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out example-rsa.key.
↪pem

# 3072 bit RSA
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:3072 -out example-rsa.key.
↪pem
```

8.1.2 ECDSA Keys

Most browsers support `secp256r1 (P-256)` and `secp384r1 (P-384)` curves. Use the following command to generate EC keys with `openssl`:

```
# P-256 EC key
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out example-ec.key.
↪pem

# P-384 EC key
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -out example-ec.key.
↪pem
```

8.1.3 Certificate Signing Request

The openssl `req` utility can be used to generate certificate signing requests suitable for `certhub`. Note that *Let's Encrypt* ignores anything in the CSR except `CN`, `subjectAltName` and the OCSP stapling tls feature flag if present. Adapt the following example to generate a CSR from the command line without having to craft a openssl.cnf file.

```
$ openssl req -new -subj "/CN=example.com" \
      -addext "subjectAltName = DNS:example.com,DNS:www.example.com" \
      -addext "basicConstraints = CA:FALSE" \
      -addext "keyUsage = nonRepudiation, digitalSignature, keyEncipherment" \
      -addext "tlsfeature = status_request" \ # Remove this line if your TLS server␣
↪is not configured for OCSP.
      -key example-ec.key.pem -out example-ec.csr.pem
```

In order to inspect any CSR, use the `-text` and `-noout` flags:

```
$ openssl req -text -noout -in example-ec.csr.pem
```

8.2 DNS Zone Setup

When using DNS-01 the *ACME Client* requires access to create and delete `TXT` records on the `_acme-challenge` subdomain of the target domain. In order to reduce risk of compromise of the main DNS zone, it is necessary to serve challenges from a different DNS zone with separate credentials, or even a dedicated DNS server.

In order to serve the challenge from a different zone, it is necessary to either **delegate** the `_acme-challenge` subdomain to another DNS server using `NS` records or to **alias** the subdomain into a dedicated zone using `CNAME` records.

8.2.1 Example setup

Services in the following domains should be protected using Let's Encrypt certificates: `www.example.com`, `example.com`.

Note, many public DNS providers do only support privilege separation on a per-domain level. Thus subdomains cannot be managed from a different account. In this case it is recommended to simply host challenge zones using a different public DNS provider. It is recommended to choose one which is supported well by the *ACME Client* in use.

As an alternative to public DNS providers, there is the option to run a dedicated stripped down non-recursive DNS server only hosting challenge zones.

8.2.2 Delegation

Assuming that a dedicated DNS service reachable at `acme-ns1.example.net` is hosting `_acme-challenge` zones. The service needs to host one `_acme-challenge` zone for each target domain. Thus if a certificate should be requested containing `example.com` and `www.example.com`, then the DNS service needs to host two zones. I.e., `_acme-challenge.example.com` and `_acme-challenge.www.example.com`.

In that case the following DNS records need to be added to the main zone:

```
_acme-challenge.www.example.com. IN NS acme-ns1.example.net
_acme-challenge.example.com.     IN NS acme-ns1.example.net
```

8.2.3 Aliasing

Assuming that there is a DNS zone `auth.example.net` dedicated to host ACME challenges. One or more DNS label(s) needs to be choosen in the dedicated DNS zone to host the `TXT` records. Note that there is no strict rule on how labels need to be named. In general it is recommended that records in a label are only updated by one *ACME client* at a time.

The following DNS records need to be added to the main zone if the label `www-example-com` should be used to serve `TXT` records inside `auth.example.com`.

```
_acme-challenge.www.example.com. IN CNAME www-example-com.auth.example.com
_acme-challenge.example.com.     IN CNAME www-example-com.auth.example.com
```

Note: Some *ACME clients* require advanced configuration to support `CNAME` records. Otherwise they will attempt to update records on the main zone.

8.2.4 Further Reading

See also:

- EFF - A Technical Deep Dive: Securing the Automation of ACME DNS Challenge Validation
- GitHub - joohoi/acme-dns

8.3 Certificates for Internal Services

For some sites it is desirable that they do not leak into the surface web. E.g., staging servers for client projects or internal applications, devices and appliances. All certificates which are issued by Let's Encrypt are recorded in the Certificate Transparency Logs.

CT Logs are a popular reconnaissance tool among security analysts, since they can be parsed easily with automated tools on large scale.

In order to prevent leaking information via CT Logs, the following measures are appropriate: Use wildcard certificates and a separate domain.

8.3.1 Wildcard Certificates

Wildcard certificates can be issued for exactly one level of subdomains. E.g., a certificate containing the SAN `*.example.com` is valid for `my-crm.example.com` but neither for `example.com` nor for `crm.apps.example.com`.

Thus it is recommended to plan with a flat subdomain structure, especially if subdomains are to be generated in an automated way.

Note that there is no need to reuse one pair of key/certificate for all services. It is completely possible to issue and deploy distinct certificates for the same wildcard domain to different hosts, as long as the rate limits are adhered to.

8.3.2 Separate Domain

Instead of using the main domain which is known to the general public, a dedicated domain can be registered and used for internal purposes. This also simplifies setup of DNS `CAA` records. E.g., the `CAA` on a dedicated domain can be restricted to wildcard certificates only.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index