# Enough Documentation

***Release 2.0.3***

**Enough community**

**Jan 04, 2021**

# Infrastructure guide


**1 Introduction** **3**
- 1.1 Requirements . . . 3
- 1.2 Quick start . . . 4

**2 Using Enough** **5**
- 2.1 Requirements . . . 5
- 2.2 Installation . . . 6
- 2.3 Upgrade . . . 6
- 2.4 Create the DNS name server . . . 6
- 2.5 Create or update a service . . . 7
- 2.6 Restore a service . . . 7
- 2.7 Infrastructure services and access . . . 8
- 2.8 Low level commands . . . 12

**3 Release Notes** **15**
- 3.1 2.1.14 . . . 15
- 3.2 2.1.13 . . . 15
- 3.3 2.1.12 . . . 15

**4 Services** **17**
- 4.1 Nextcloud . . . 17
- 4.2 Forum . . . 17
- 4.3 Mattermost . . . 17
- 4.4 Etherpad . . . 17
- 4.5 Weblate . . . 18
- 4.6 GitLab . . . 18
- 4.7 Hugo . . . 18
- 4.8 Wekan . . . 18
- 4.9 openedX . . . 18
- 4.10 SecureDrop . . . 18
- 4.11 DNS . . . 19
- 4.12 VPN . . . 19
- 4.13 SMTP server . . . 20
- 4.14 Psono . . . 21
- 4.15 Intrusion Detection System . . . 21
- 4.16 Monitoring . . . 21
- 4.17 Backups . . . 22

4.18 Jitsi . . . 22
4.19 WordPress . . . 22

**5 Community** **25**
5.1 Documentation . . . 25
5.2 Domain . . . 25
5.3 Hosting and infrastructure . . . 26
5.4 Extending *enough.community* . . . 27
5.5 Contribute . . . 29
5.6 Release management . . . 34
5.7 Team . . . 35
5.8 Funding . . . 36

The Enough infrastructure are a set of services (Forum, chat, shared storage, monitoring, backups, IDS, DNS, etc.) to support the work of the Enough community. It is maintained by *individuals* organized horizontally. Anyone is welcome to contribute: learn more in the *contribution guide*.

Enough is also suitable for organizations with a focus on human right defenders and journalists. It can be used to create and maintain an independant infrastructure.

CHAPTER 1

# Introduction

Enough is a platform for journalists, sources and human rights defenders to communicate privately and securely. It provides the following services:

- Nextcloud, a suite of client-server software for creating and using file hosting services.
- Discourse, a discussion platform built for the next decade of the Internet. Use it as a mailing list, discussion forum, long-form chat room, and more!
- Mattermost, a flexible messaging platform that enables secure team collaboration.
- Hugo, a static web site generator.
- Weblate, a libre web-based translation tool with tight version control integration. It provides two user interfaces, propagation of translations across components, quality checks and automatic linking to source files.
- Wekan, a kanban board which allows a card-based task and to-do management.
- Etherpad, a highly customizable online editor providing collaborative editing in really real-time.
- GitLab, a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and continuous integration/continuous deployment pipeline features.
- OpenVPN, that implements virtual private network (VPN) techniques to create secure point-to-point or site-to-site connections in routed or bridged configurations and remote access facilities

The *enough* CLI controls an OpenStack based infrastructure and the services that run on top of it, with Ansible.

## 1.1 Requirements

- An account on a supported OpenStack provider:
  - A `Public cloud` project at OVH. A Private network must be created for the project.
  - A Fuga account.
- The *clouds.yml* credentials for:

  - OVH, found in horizon
  - Fuga, found in the Team Credentials tab

## 1.2 Quick start

- Install Docker.
- Copy *clouds.yml* in *~/.enough/myname.d.enough.community/inventory/group_vars/all/clouds.yml* and edit to add *password:* next to *auth_url:*, under the *production:* name. For instance:

```
---
openstack_provider: fuga
clouds:
  production:
    auth:
      auth_url: "https://identity.api.ams.fuga.cloud:443/v3"
      user_id: "ef5cae1b61b8424594a6ddf94a28381c"
      password: "lDk9vOLIXFW09oWcuQEiq0sjB4cV"
      user_domain_id: "b919e18e477a889bf89f89e9d9"
      project_domain_id: "b919e186cb07a889bf89f89e9d9"
      project_id: "25481e67871b4de39ae63fa2008029"
    region_name: "ams"
    interface: "public"
    identity_api_version: 3
```

- Add the `enough` CLI to `~/.bashrc`:

```
eval "$(docker run --rm enoughcommunity/enough:latest install)"
```

- Create the `Nextcloud` service with:

```
$ enough --domain myname.d.enough.community service create cloud
```

**Note:** If the command fails, because of a network failure or any other reason, it is safe to run it again. It is idempotent.

- Login `https://cloud.myname.d.enough.community` with user `admin` password `mynextcloud`
- Display the hosts that were created and the services they run:

```
$ enough --domain myname.d.enough.community info
bind-host ip=51.168.48.253 port=22
   bind
cloud-host ip=51.68.77.181 port=22
   cloud   https://cloud.myname.d.enough.community
           nextcloud_admin_user=admin
           enough_nextcloud_version=19
           nextcloud_admin_pass=*****
```

CHAPTER 2

# Using Enough

## 2.1 Requirements

### 2.1.1 Technical

- The `clouds.yml` credentials for either OVH or Fuga.
- The current code base requires:
  - The Orchestration API available: *Heat*, at least `heat_template_version: 2016-10-14/newton`.
  - A public IPv4 per virtual machine (not a *floating IP* but a *direct IP*). IPv6 isn't supported yet.
  - In order to deploy every available services: 15 virtual machines.
  - The flavors must be provided with an attached root disk by default (not an explicit block storage) most of the virtual machines use 2Go RAM, some hosts/tests require 4Go or 8Go RAM.
  - A Debian GNU/Linux stable image.

### 2.1.2 Knowledge

Enough is based on Ansible, OpenStack and Debian GNU/Linux. Debian GNU/Linux based virtual machines are created on an OpenStack tenant and provisioned with Ansible playbooks.

Each service adds more components such as Let's encrypt, Docker or Snap. And they also implement concepts such as Virtual Private Network, Reverse Proxy or Certificate Authority.

If all goes well, Enough can be used even if the user knows nothing more than what is in this guide. When something goes wrong, fixing the problem or understanding the cause will require intimate knowledge about all of the above. If that happens, it is best to start a discussion in the forum and ask for help.

## 2.2 Installation

- Install Docker.
- Copy *clouds.yml* in *~/.enough/example.com/inventory/group_vars/all/clouds.yml* and edit to add *password:* next to *auth_url:* under the *production:* name.
- Add the `enough` CLI to `~/.bashrc`:

```
eval "$(docker run --rm enoughcommunity/enough:latest install)"
```

- Verify that it works:

```
enough --version
```

## 2.3 Upgrade

To upgrade to the latest Enough version:

```
$ docker pull enoughcommunity/enough:latest
$ eval "$(docker run --rm enoughcommunity/enough:latest install)"
```

## 2.4 Create the DNS name server

Assuming the domain name (`example.com` for example) is registered, it must be delegated to a dedicated name server before any service can be created by Enough:

```
enough --domain example.com service create bind
```

Upon successfull completion, a machine named `bind-host` exists and its public IP must be used as a GLUE record.

```
$ enough --domain example.com openstack server list
+--------------+--------------+--------+----------------------+-----------+--------+
| ID           | Name         | Status | Networks             | Image     | Flavor |
+--------------+--------------+--------+----------------------+-----------+--------+
| 2b9a1bda-c2c0 | bind-host    | ACTIVE | Ext-Net=51.178.60.121 | Debian 10 | s1-2   |
+--------------+--------------+--------+----------------------+-----------+--------+
```

To verify the DNS running at bind-host works as expected:

```
$ dig @ip-of-the-bind-host bind.example.com
$ enough --domain example.com ssh bind-host
```

It can then be used to instruct the registrar of `example.com` to delegate all domain name resolutions to this IP. The exact method for performing this DNS delegation depends on the registrar (Gandi is different from OVH, etc.). But it needs to be done only once.

---

**Note:** It will take time for the delegation to be effective. It can be as quick as one hour but delays from 24h to 72h are not uncommon.

---

To verify the delegation is effective:

```
getent hosts bind.example.com
```

## 2.5 Create or update a service

The following services are available:

- *bind* for DNS server at `bind.examples.com`
- *icinga* for monitoring at `icinga.example.com`.
- *postfix* for SMTP server at `postfix.example.com`.
- *OpenVPN*, for VPN at `openvpn.example.com`
- *wazuh* for Intrusion Detection System at `wazuh.example.com`.
- *chat*, for instant messaging at `chat.example.com`
- *cloud*, for file sharing at `cloud.example.com`
- `forum`, for discussions and mailing lists at `forum.example.com`
- `packages`, a static web service at `packages.example.com`
- `pad`, for collaborative note taking at `pad.example.com`
- *Weblate*, for online translations at `weblate.example.com`
- *WordPress*, for CMS at `wordpress.example.com`
- *openedX*, for MOOC platform at `openedx.example.com`
- `website`, for static websites at `website.example.com`
- `wekan`, for kanban at `wekan.example.com`
- *gitlab*, for software development at `lab.example.com`
- `api`, for *Enough development* at `api.example.com`
- *Jitsi*, for video conferencing at `jitsi.example.com`
- *Psono*, for password management at `psono.example.com`

As an example, the cloud service can be created as follows:

```
enough --domain example.com service create cloud
```

**Note:** If the command fails, because of a network failure or any other reason, it is safe to run it again. It is idempotent.

When it completes successfully, it is possible to login `https://cloud.example.com` with user `admin` and password `mynextcloud`.

## 2.6 Restore a service

Stateless services such as *bind* do not need backup: they can be rebuilt from scratch if the machine hosting them fails. For instance, if *bind-host* is lost:

```
$ enough --domain example.com host create bind-host
$ enough --domain example.com playbook
```

However, most services such as *file sharing* and *translations* rely on persistent information that are located in a encrypted volume attached to the machine. A daily *backup* is made in case a file is inadvertendly lost.

## 2.7 Infrastructure services and access

### 2.7.1 Networks

By default all hosts are connected to two networks: one with a public IP and the other with a private IP. This can be changed by setting the *network_internal_only* variable in *~/.enough/example.com/inventory/group_vars/all/network.yml*, using this example.

The default can also be changed for a given host (for instance *weblate-host*) by setting the desired value in the *~/.enough/example.com/inventory/host_vars/weblate-host/network.yml* file.

### 2.7.2 VPN

A VPN can optionally be installed for clients to access hosts that do not have public IPs.

A host with a public IP must be chosen to deploy the VPN. For instance *bind-host* by adding the following to *~/.enough/example.com/inventory/services.yml*:

```
openvpn-service-group:
  hosts:
    bind-host:
```

It can then be created with:

```
enough --domain example.com service create openvpn
```

The certificates for clients to connect to the VPN will be created from the list in the *openvpn_active_clients* variable in *~/.enough/example.com/inventory/group_vars/all/openvpn.yml*, using this example.

For each name in the *openvpn_active_clients* list, a *.tar.gz* file will be created in the *~/.enough/example.com/openvpn/* directory. For instance, for

```
---
openvpn_active_clients:
 - loic
```

The file *~/.enough/example.com/openvpn/loic.tar.gz* will be created and contains OpenVPN credentials. The specific instructions to use them depends on the client.

### 2.7.3 Certificates

By default certificates are obtained from Let's Encrypt. But if a host is not publicly accessible, it can be configured to obtain a certificate from a certificate authority dedicated to the Enough instance. The default for *certificate_authority* should be set in *~/.enough/example.com/inventory/group_vars/all/certificate.yml*, using this example.

The default can also be changed for a given host (for instance *weblate-host*) by setting the desired value in the *~/.enough/example.com/inventory/host_vars/weblate-host/network.yml* file.

### 2.7.4 Attached volumes

#### Provisioning

A volume can be created and attached to the host. It can be resized at a later time, when more space is needed. For instance, before creating *weblate-host*, the desired volume size and name can be set in the *~/.enough/example.com/inventory/host_vars/weblate-host/provision.yml* file like so:

```
---
openstack_volumes:
  - name: weblate-volume
    size: 10
```

#### Encrypting and Mounting

The volume can then be encrypted, formatted and mounted by specifying the mount point in the *encrypted_device_mount_point* variable like so:

```
---
openstack_volumes:
  - name: weblate-volume
    size: 10
encrypted_device_mount_point: /srv
```

By default Docker or Snap will be set to reside in the *encrypted_device_mount_point* directory so that the data it contains is encrypted. It can be disabled with the *encrypted_volume_for_docker* and *encrypted_volume_for_snap* variables like so:

```
---
openstack_volumes:
  - name: weblate-volume
    size: 10
encrypted_device_mount_point: /srv
encrypted_volume_for_docker: false
encrypted_volume_for_snap: false
```

#### Resizing

The size of a volume can be increased (but never decreased) by modifying the value from (for instance) 10GB

```
---
openstack_volumes:
  - name: weblate-volume
    size: 10
```

to 20GB

```
---
openstack_volumes:
  - name: weblate-volume
    size: 20
```

The resize operation is done with the following command (the host will be rebooted). If the volume already has the desired size, the command will do nothing.

```
$ enough --domain example.com volume resize weblate-host weblate-volume
```

If the volume is mounted as an encrypted partition, it should then be extended to use the additional disk space. There is no need to unmount the partition.

```
$ enough --domain example.com ssh weblate-host -- sudo cryptsetup resize --key-file=/
↪etc/cryptsetup/keyfile spare
$ enough --domain example.com ssh weblate-host -- sudo resize2fs /dev/mapper/spare
```

### 2.7.5 Services

The following services are always available:

- *bind* for DNS server at `bind.examples.com`
- security groups for *firewall*.

### 2.7.6 Background tasks

- *Volumes and hosts backups*.
- Unattended upgrades.
- Tracking changes in /etc/ for each machine.

### 2.7.7 Access

The SSH public keys found in files matching `authorized_keys_globs` are installed on every machine.

```
---
authorized_keys_globs:
  - ssh_keys/dachary.pub
  - ssh_keys/glen.pub
```

### 2.7.8 Restore a service from a backup

To restore the volume attached to a service from a designated backup:

```
$ enough --domain example.com openstack volume snapshot list
...
| 6b75f34e | 2020-04-12-cloud-volume | None | available | 100 |
...
$ enough --domain example.com backup restore 2020-04-12-cloud-volume
```

In this example, the restoration is done as follows:

- The *cloud service* is created, if it does not already exist.
- The machine (`cloud-host`) attached to the volume (`cloud-volume`) is stopped. The volume is detached and deleted.
- A new volume `cloud-volume` is created from the `2020-04-12-cloud-volume` backup and attached to `cloud-host`.
- The machine (`cloud-host`) is restarted.

### 2.7.9 Create a clone of a service from a backup

It is convenient to create a clone of an existing service based on a backup for:

- testing and experimenting without disrupting production
- verify an upgrade won't loose any data
- teaching
- etc.

```
$ enough --domain example.com openstack volume snapshot list
...
| 6b75f34e | 2020-04-12-cloud-volume | None | available | 100 |
...
$ enough --domain example.com backup restore \
         --target-domain test.d.enough.community \
         2020-04-12-cloud-volume
```

Once the service is cloned, it will be available at `https://cloud.test.d.enough.community`. In this example, the cloning is done as follows:

- A dedicated OpenStack region is used to restore the backup

---

**Note:** The OpenStack region where the backup is restored is in the *clone* section of the *~/.enough/example.com/inventory/group_vars/all/clouds.yml* file and it can be modified if the default is not suitable.

---

- A volume is created from the `2020-04-12-cloud-volume` snapshot
- The *cloud service* is created (in the region dedicated to restoring the backup) as well as all the services it depends on, if they do not already exist. Including the *DNS server*.
- The `test.d.enough.community` domain is delegated to the *DNS server* located in the OpenStack region where the backup was restored so that `https://cloud.test.d.enough.community` resolves to the newly created *cloud service*.

It is possible restore the service step by step with the following commands:

```
$ enough --domain example.com backup clone volume \
         --target-domain test.d.enough.community 2020-07-29-cloud-volume
$ enough --domain test.d.enough.community service create cloud
$ enough --domain test.d.enough.community backup restore 2020-07-29-cloud-volume
```

### 2.7.10 Restoring a service that requires a VPN

If the service restored in a clone requires a VPN (that is if it runs on an private IP), a new VPN must be setup before the user can access it.

If the service is cloned with:

```
$ enough --domain example.com backup restore \
         --target-domain test.d.enough.community \
         2020-04-12-cloud-volume
```

The credentials to connect to the VPN of the clone are found in the *~/.enough/test.d.enough.community/openvpn* directory (for instance *~/.enough/test.d.enough.community/openvpn/loic.tar.gz*).

**Note:** Although the *loic.tar.gz* file has the same name as in the original, it will connect to a the VPN server in the clone. Care must be taken to **not** override credentials that existed before the cloning operation.

The subnet of internal network of the clone is hardcoded in *.enough/test.d.enough.community/inventory/group_vars/all/internal_network.*

## 2.8 Low level commands

The following are not useful if only relying on the `service` command above. They can however be helpful to run Ansible or OpenStack manually.

### 2.8.1 Adding hosts

The hosts (OpenStack virtual machines) are created automatically when a service is provided. It is however possible to create a new host or destroy an existing one.

The first step is to edit `~/.enough/example.com/inventory/all.yml` and add the name of the new host:

```
---
all-hosts:
 hosts:
  my-host:
  bind-host:
  forum-host:
  ...
```

Creating a new host:

```
enough --domain example.com host create my-host
```

SSH to a host:

```
enough --domain example.com ssh my-host
```

### 2.8.2 Removing hosts

Every host is known to `icinga`, `bind` and `wazuh` and it should be deleted from these services before being removed.

- Add the host to the `deleted-hosts` group in `~/.enough/example.com/inventory/all.yml`:

```
---
deleted-hosts:
  hosts:
    some-host:
```

- Run the playbook:

```
enough --domain example.com playbook
```

- Physically delete the host

```
enough --domain example.com host delete my-host
```

### 2.8.3 Running openstack

The openstack CLI can be used as follows:

```
$ enough --domain example.com openstack -- help
```

Which is exactly equivalent to:

```
$ OS_CLIENT_CONFIG_FILE=~/.enough/example.com/inventory/group_vars/all/clouds.yml \
  openstack --os-cloud production help
```

### 2.8.4 Running ansible-playbook

The ansible-playbook CLI can be used as follows:

```
$ enough --domain example.com playbook -- --limit localhost,icinga-host \
  --private-key ~/.enough/example.com/infrastructure_key \
  ~/.enough/example.com/enough-playbook.yml
```

It implicitly uses the following inventories (via multiple **–inventory** options), in order (the last inventory listed has precedence):

- ~/.enough/example.com/inventory
- built in Enough inventory

CHAPTER 3

# Release Notes

## 3.1 2.1.14

### 3.1.1 postfix

- Fixes a bug blocking all outgoing mails on the relay.

## 3.2 2.1.13

### 3.2.1 gitlab

- Add missing dependencies (debops.libvirt*) that would fail when trying to deploy a CI runner.

## 3.3 2.1.12

### 3.3.1 icinga

The icinga client address was *hostvars[inventory_hostname]['ansible_host']* prior to 2.1.12. It now is *icinga_client_address* which defaults to *hostvars[inventory_hostname]['ansible_host']*. It can be used to resolve the following problem:

- The icinga master has a private IP and no public IP
- The icinga master goes through a router with a public IP
- The icinga client has a public IP which is the default for *icinga_client_address*
- The icinga master tries to ping the icinga client public IP but fails because the firewall of the client does not allow ICMP from the router public IP

The *icinga_client_address* of the client is set to the internal IP instead of the public IP. The ping will succeed because the firewall allows ICMP from any host connected to the internal network.

### 3.3.2 Development

- Added basic support for running tests with libvirt instead of OpenStack.

CHAPTER 4

# Services

## 4.1 Nextcloud

Nextcloud is available at *cloud.example.com*. The user with administrative rights, the Nextcloud version and other variables are documented in this file and can be modified in the *~/.enough/example.com/inventory/host_vars/cloud-host/nextcloud.yml* file.

## 4.2 Forum

Discourse is available at *forum.example.com*.

## 4.3 Mattermost

Mattermost is available at *chat.example.com*.

## 4.4 Etherpad

Etherpad is available at *pad.example.com*. The user with administrative rights is *admin*. Its password can be set as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/pad-service-group.yml* file.

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host pad-host pad
```

## 4.5 Weblate

Weblate is available at *weblate.example.com*. The user with administrative rights and the contact email are defined as documented in this file and can be modified in the *~/.enough/example.com/inventory/host_vars/weblate-host/secrets.yml* file.

## 4.6 GitLab

GitLab is available at *lab.example.com*. The user with administrative rights is *root*. Its password can be set as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/gitlab/gitlab.yml* file.

## 4.7 Hugo

Hugo is available at *www.example.com*.

## 4.8 Wekan

Wekan is available at *wekan.example.com*. The user with administrative rights is *admin*. Its password can be set as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/wekan-service-group.yml* file.

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host website-host wekan
```

## 4.9 openedX

openedX is available at *openedx.example.com*. The user with administrative rights and the contact email are defined as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/openedx-service-group.yml* file.

## 4.10 SecureDrop

SecureDrop is only available via Tor. The administrative user, its password and TOTP can be set as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/securedrop-service-group.yml* file.

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host website-host securedrop
```

### 4.10.1 URLs

The URL of the source and journalist interfaces can be retrieved with:

```
$ enough --domain example.com ssh securedrop-host cat /var/lib/tor/services/source/
↪hostname /var/lib/tor/services/journalist/hostname
```

The URL of the journalist interface requires an authentication token and must be copied in the torrc file.

## 4.11 DNS

### 4.11.1 Records

When a new host is created (for instance with *enough –domain example.com host create cloud-host*) the names *cloud-host.example.com* and *cloud.example.com* are added to the DNS.

The *bind_zone_records* variable is inserted in the *example.com* zone declaration verbatim (see the BIND documentation for information). It can be set in *~/.enough/example.com/inventory/host_vars/bind-host/zone.yml* like so:

```
bind_zone_records: |
     imap 1800 IN CNAME access.mail.gandi.net.
     pop 1800 IN CNAME access.mail.gandi.net.
     smtp 1800 IN CNAME relay.mail.gandi.net.

     @ 1800 IN MX 50 fb.mail.gandi.net.
     @ 1800 IN MX 10 spool.mail.gandi.net.
```

### 4.11.2 Host Resolver

The resolver of all hosts (in */etc/resolv.conf*) is set with the IP of the DNS server that was *created to bootstrap Enough*. It is used to resolve the host names in the Enough domain (for instance *example.com* or *cloud.example.com*) and all other domain names (for instance *gnu.org* or *fsf.org*).

### 4.11.3 VPN Resolver

When a client connects to the *VPN*, its resolver is set to the Enough DNS server.

---

**Note:** Using the Enough DNS instead of the DNS of an internet service provider bypasses rewrites of DNS entries (imposed by by the state in some cases).

---

## 4.12 VPN

Enough hosts can be connected to a public network (with public IP addresses) and an internal network (with private IP addresses. When a host is not connected to the public network, it can only be accessed in two ways:

- By connecting to a host connected to both the public network and the internal network.
- By connecting to the VPN (which is running on a host connected to both the public network and the internal network).

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host bind-host openvpn
```

### 4.12.1 VPN Server configuration

The OpenVPN server is configured with variables (see the documentation).

### 4.12.2 VPN subnet

The default subnet used by the internal network and routed by the VPN on the client machine is defined in a configuration file that may be modified in case it conflicts with an already used subnet.

### 4.12.3 VPN Clients creation

The certificates for clients to connect to the VPN will be created from the list in the *openvpn_active_clients* variable in *~/.enough/example.com/inventory/group_vars/all/openvpn.yml*, using this example.

For each name in the *openvpn_active_clients* list, a *.tar.gz* file will be created in the *~/.enough/example.com/openvpn/* directory. For instance, for

```
---
openvpn_active_clients:
 - loic
 - glen
```

After running *enough –domain example.com playbook*, the files *~/.enough/example.com/openvpn/loic.tar.gz* and *~/.enough/example.com/openvpn/glen.tar.gz* will be created and will contain the credentials.

On Debian GNU/Linux the *.tar.gz* can be extracted in a *vpn* directory and the *.conf* file it contains imported using the *Network => VPN* system settings.

### 4.12.4 VPN Clients retirement

When a client should no longer be allowed in the VPN, it must be added in the *openvpn_retired_clients* list, using this example.

## 4.13 SMTP server

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host postfix-host postfix
```

A SMTP server is running on each host. A service running on *some-host.example.com* can use the SMTP server as follows:

- Address: some-host.example.com
- Port: 25
- Authentication: No
- SSL/TLS: No

It is not possible (and it would not be secure) for services running on another host (*other-host.example.com* for instance) to use this SMTP server.

### 4.13.1 Encryption

Outgoing mails are encrypted if the recipient has a known GPG public key. The list of GPG public keys must be provided in the *~/.enough/example.com/inventory/host_vars/postfix-host/gpg.yml* file like so:

```
---
postfix_gpg_keys:
 - "{{ '~/.enough/example.com/gpg/*.asc' | expanduser }}"
```

## 4.14 Psono

Psono is documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/psono-service-group.yml* file.

The service is created with:

```
$ enough --domain example.com service create psono
```

## 4.15 Intrusion Detection System

The Wazuh Intrusion Detection System watches over all hosts and will report problems to the *ids@example.com* mail address.

The wazuh API user and password must be created to allow the agents to register on the server. For instance:

```
$ cat ~/.enough/example.com/group_vars/all/wazuh.yml
---
wazuh_mailto: contact@enough.community
wazuh_email_from: contact@enough.community
wazuh_api_username: apiuser
wazuh_api_password: .S3cur3Pa75w0rd-#
```

---

**Note:** The password must obey the wazuh requirements to be valid. The command line *apg -n 1 -M SNCL -m 8 -x 16* will generate a suitable password.

---

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host wazuh-host wazuh
```

## 4.16 Monitoring

The Icinga Monitoring System watches over all hosts (disk space, load average, security updates, etc.). In addition services may add specific monitoring probes such as loading a web page and verifying its content is valid.

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host icinga-host icinga
```

The Icinga web interface is at *icinga.example.com*. The user name and password with administrator rights must be defined in *~/.enough/example.com/inventory/host_vars/icinga-host/icinga-secrets.yml* with variables documented in this file

Problems found by Icinga will be notified via email to the address defined in *~/.enough/example.com/inventory/host_vars/icinga-host/mail.yml* with a variable doumented in this file.

The Icinga master pings the icinga client using *icinga_client_address* as found in this file.

## 4.17 Backups

Persistent data is placed in *encrypted volumes* otherwise it may be deleted at any moment, when the host fails. A daily backup of all volumes is done on the host designated to host backups (for instance *bind-host*) when the service is created as follows:

```
$ enough --domain example.com service create --host bind-host backup
```

The number of backups is defined with the *backup_retention_days* variable as documented in this file and can be set in *~/.enough/example.com/inventory/group_vars/backup-service-group.yml* like so:

```
---
backup_retention_days: 7
```

**Note:** If the quota for volume snapshots displayed by *enough –domain example.com quota show* is too low, a support ticket should be opened with the cloud provider to increase it.

A volume backup can be used to *restore a service* in the state it was at the time of the backup.

The volumes are replicated three times and their data cannot be lost because of a hardware failure.

## 4.18 Jitsi

Jitsi is documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/jitsi-service-group.yml* file.

The service is created with:

```
$ enough --domain example.com service create jitsi
```

## 4.19 WordPress

WordPress is available at *wordpress.example.com*. The user with administrative rights and the contact email are defined as documented in this file and can be modified in the *~/.enough/example.com/inventory/group_vars/wordpress-service-group.yml* file.

The service is created on the host specified by the *–host* argument:

```
$ enough --domain example.com service create --host wordpress-host wordpress
```

CHAPTER 5

# Community

## 5.1 Documentation

The documentation for *enough.community* is published at https://enough-community.readthedocs.io

## 5.2 Domain

The *enough.community* domain name is registered at Gandi under the user ieNua8ja.

After the *bind-host* virtual machine is created, click on *Glue record management* in the Gandi web interface and set ns1 to the IP, i.e. 51.68.79.8 and wait a few minutes. Click on *Update DNS* and set the *DNS1* server to ns1.enough.community and click on *Add Gandi's secondary nameserver* which should add a new entry in DNS2: it will automatically act as a secondary DNS.

The *bind-host* virtual machine should be initialized before any other because everything depends on it.

### 5.2.1 Mail

The admin mail is hosted at Gandi and is used as the primary contact for all *enough.community* resources (hosting etc.). In case a password is lost this is the mail receiving the link to reset the password etc.

### 5.2.2 Zones

#### enough.community

The *enough.community* zone is managed on a dedicated virtual machine *ns1.enough.community*. It is generated via the bind playbook.

- The port udp/53 is open to all but recursion is only allowed for IPs of the enough.community VMs
- An **A** record is created for all existing VM names

- A **CNAME** record is created for all VM names without the *-host* suffix
- The *SPF* **TXT** record help *send mail* successfully.

#### test.enough.community and d.enough.community

They can be updated locally by the *debian* user via `nsupdate`. Example:

```
- E - debian@bind-host:~$ nsupdate <<EOF
server localhost
zone test.enough.community
update add bling.test.enough.community. 1800 TXT "Updated by nsupdate"
show
send
quit
EOF
```

## 5.3 Hosting and infrastructure

### 5.3.1 OpenStack at OVH

All virtual machines are in the OVH OpenStack cloud. The OVH account is **ce188933-ovh** (login via https://www.ovh.com/auth/) and is bound to the enough.community admin mail.

The following OVH projects have been defined:

---

**Note:** The OVH user is the paying customer and the OVH projects are completely isolated from each other. The OVH interface allows to create OpenStack tenants in a given project. An OpenStack tenant only has access to the OVH project in which it has been created. A tenant has access to all the regions.

---

- **OVH Project: Contributors**
  - Region **BHS3** & **UK1**: used for testing by Loïc Dachary
  - Region **SBG5**: used for testing by Pierre-Louis Bonicoli
  - Region **WAW1**: used for testing by François Poulain
  - Region **GRA5** & **DE1**: used for testing by Kim Minh Kaplan
- **OVH Project: CI**
  - Region **DE1**: GitLab runner
  - Region **GRA5** & **UK1**: used for testing by nesousx
- **OVH Project: Production**
  - Region **GRA5**: Instances running services in the *enough.community* domain
- Login as a customer
- OpenStack OVH management

### 5.3.2 OpenStack at Fuga

The following teams are defined in Fuga:

- Team **pimthepoi** used for testing by pimpthepoi

### 5.3.3 Security groups

The firewall to all machines is based on openstack security groups. There is one security group per VM.

### 5.3.4 VM naming conventions

All VMs names end with *-host* because it makes them easier to grep.

### 5.3.5 Global account name

The *debian* account exists on all VMs and is used by all for configuration and debug.

## 5.4 Extending *enough.community*

After *getting started*, you may want to add some host/service in *enough.community infrastructure*.

### 5.4.1 Overview

The *enough.community* infrastructure is developed and deployed using ansible. The main playbook is *enough-community-playbook.yml*.

Each playbook (for instance *playbooks/weblate/weblate-playbook.yml*) is a *scenario*, i.e. a set of hosts and a test playbook to get a minimal infrastructure setup allowing to validate the service playbook. As an example, the weblate scenario found in *playbooks/weblate/* defines the following hosts:

- a postfix master
- a bind server
- an icinga server
- the weblate host

With these, the *weblate* scenario is able to test the weblate deployment, its monitoring, its ability to send emails, etc.

Some generally useful playbooks are grouped in the *misc* scenario.

The presence of a *bind-host* in most scenarios allows to spoof all records from *enough.community* domain during the tests (and could spoof any other domain). This disallow external requests like e.g. ACME challenge for TLS Certificates. To overcome this limitation, the domain of the scenario is defined in a one-time testing subdomain when a *bind-host* is used by the scenario and the variable *letsencrypt_staging* has been defined (meaning that we should use the "fake Let's Encrypt unlimited testing infrastructure").

The *preprod* scenario aggregates all *enough.community* playbooks and should be used to very they fit together as expected.

### 5.4.2 Adding a scenario

Let's start with a *dummy* scenario.

#### Copying from an existing scenario

It is recommended to copy an existing scenario that ressembles the one to be created.

```
cp -a playbooks/website playbooks/dummy
```

Add *dummy* on the same line as *weblate* in *tox.ini*

#### Scenario definition

Edit *playbooks/dummy/conftest.py* to:

- update the list of hosts
- update the name of the new scenario

#### Adding playbooks

The playbook is *playbooks/dummy/playbook.yml*. It should include all playbooks used for the scenario, i.e.:

- others scenarios playbooks, like *playbooks/icinga/icinga-playbook.yml* or *playbooks/postfix/postfix-playbook.yml*
- the playbook specific to this scenario, here *playbooks/icinga/dummy-playbook.yml*. This playbook may include other playbooks.
- tests specific playbooks, starting with *test*, e.g. *playbooks/icinga/test-dummy-playbook.yml*.

Once the playbooks are added to run them with:

- tox -e weblate

#### Adding tests

The purpose of the tests is mainly to detect that Ansible has deployed a functional service. See them as *functionnal and non-regression testing* to maintaining our Ansible base.

testinfra is used for this purpose. The easiest way to get started with it is to look at some existing tests. For simple testing see *playbooks/bind/tests/test_external_bind.py*. For a request based test, see e.g. *playbooks/weblate/tests/test_icingaweb.py*.

Since the tests run with virtual machines provisionned exclusively for the test, you can do whatever you want (i.e. even some destructive action).

Testing is not monitoring. You are kindly invited to setup monitoring for your services and to test via testinfra that monitoring has been setup as you wish.

#### Interaction with others scenarios

Most services rely on *DNS*, *emails* and *monitoring*. To enable them you have to add the corresponding hosts in your scenario and include their playbook in your scenario's playbook.

You will also be interested by:

- *playbooks/misc/sexy-debian-playbook.yml* for getting usefull tools,
- *playbooks/authorized_keys/authorized-keys-playbook.yml* for installing ssh keys,
- *playbooks/misc/commit_etc-playbook.yml* for committing changes to */etc/* at the end of your playbook.

#### Documentation

You are kindly invited to document your scenario in the *docs* directory where this documentation lives.

## 5.5 Contribute

This is the contribution guide to the *enough.community* infrastructure which is based on Ansible and OpenStack. If you're a seasoned Free Software contributor looking for a quick start, take a look at the list of bugs and features, otherwise keep reading.

### 5.5.1 Resources

- Repository and issue tracking: http://lab.enough.community/main/infrastructure
- Forum: https://forum.enough.community/
- Chat: https://chat.enough.community/enough/
- License: AGPLv3
- *Who's who*

### 5.5.2 Bugs and features list

Each service under the *enough.community* domain can be worked on independently and have their own integration tests. There is no need to understand how *Weblate* is deployed if you're improving *Discourse*, for instance.

### 5.5.3 Organization

All contributors are organized horizontally

- People with access to an exclusive resource must register themselves in the *team directory*

### 5.5.4 Getting started

- `git clone https://lab.enough.community/main/infrastructure`

### 5.5.5 Running tests

- Install Docker.

The `tests/run-tests.sh` script builds a docker image suitable for running the tests, with all the required dependencies, based on a Debian GNU/Linux buster. The following volumes are bind-mounted:

- ~/.enough
- ~/.ansible
- the root of the *infrastructure* repository

The working directory, in the container, is the root of the *infrastructure* repository.

#### Installing libvirt

Manually run commands similar to *playbooks/gitlab/roles/gitlab-ci/tasks/gitlab-ci.yml* (it could be a playbook running on localhost with sudo sudo ?)

#### Running tests that do not require OpenStack

- `PYTEST_ADDOPTS='-m "not openstack_integration"' tests/run-tests.sh`

#### Running tests that require OpenStack

#### Introduction

The tests running without OpenStack only cover a fraction of what Enough does. To verify that a playbook actually works, it needs to be run on a live host and tests must check that is working. For instance the tests for *Weblate* request that the *Weblate* server sends a mail and verify it reaches the postfix server.

When modifying a role or a playbook in the directory *playbooks/ABC* one is expected to add a test for the new behavior and verify it runs successfully:

- `tests/run-tests.sh tox -e ABC`

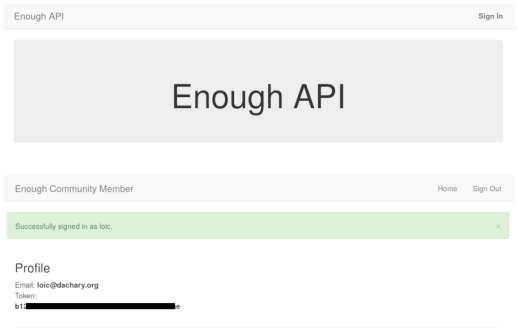
When relevant, integration tests should be created as *icinga* monitoring checks so they can be run on a regular basis in the production environment to verify it keeps working.

#### SSH key

A SSH authentication key is generated by the tests which requires OpenStack. The private key is named *infrastructure_key*, the public key is named *infrastructure_key.pub* and both are located in *.tox/ABC/.pytest_cache/d/dotenough/ABC.test/* directory.

#### Obtain an API token

Most integration tests need a publicly available DNS server. The https://api.enough.community provides a publicly available API to delegate a domain to the designated DNS server. Members of the group enough can sign-in, others can request access.




The **Token:** value displayed after signing in https://api.enough.community must be set to the `ENOUGH_API_TOKEN` environment variable.

- `ENOUGH_API_TOKEN=XXXXXXX tests/run-tests.sh tox -e bind`

### Set the OpenStack credentials using clouds.yml

Assuming you have your own OpenStack tenant or one was *provided to you*, the `clouds.yml` file must be copied to *tests/clouds.yml*. The *openstack_provider* variable must be added to the clouds.yml file:

```
---
openstack_provider: fuga
clouds:
...
```

or

```
---
openstack_provider: ovh
clouds:
...
```

It must define two cloud environment: *production* and *clone* (for backup restoration testing purposes). Here is a complete example:

```
---
openstack_provider: fuga
clouds:
  production:
    auth:
      auth_url: "https://identity.api.ams.fuga.cloud:443/v3"
      user_id: "6a79dfb7410c4884fceb23031189b"
      password: "qecOSdBAH6ZjE4M2UnZbnnWdsZihe"
      user_domain_id: "99009ec244eebb85827488bb2aed4"
      project_domain_id: "9900e2c244eebb85827488bb2aed4"
      project_id: "203e72ec8a85b9dc808719e452902"
    region_name: "ams"
    interface: "public"
    identity_api_version: 3
  clone:
    auth:
      auth_url: "https://identity.api.ams.fuga.cloud:443/v3"
      user_id: "3b40cf2cb71b4bdc95c009347445f"
      password: "RBX0S2BdXWlBztUKkPWcAfnNFSNNj"
      user_domain_id: "de844dabe43948cb87ed24e2d5c438a9"
```

(continues on next page)

(continued from previous page)

```
      project_domain_id: "de8abe43948cb87ed24e2d5c438a9"
      project_id: "82cb2f62a70f5928e3a4686622e39"
    region_name: "ams"
    interface: "public"
    identity_api_version: 3
```

#### Running

- `tests/run-tests.sh tox -e <service name>`

**Note:** If the command fails, because of a network failure or any other reason, it is safe to run it again. It is idempotent and will re-use the environment from the failed test.

The list of service names (i.e. tox test environments) is in the *tox.ini* file. It is possible to skip some steps to speed up test debugging:

```
$ tox -e bind -- --help playbooks
...
custom options:
  --enough-no-create    Do not run the create step
  --enough-no-tests     Do not run the tests step
  --enough-no-destroy   Do not run the destroy step
...
$ tests/run-tests.sh tox -e authorized_keys -- --enough-no-destroy playbooks/
↪authorized_keys/tests
```

The domain name used for testing is in *.pytest_cache/d/dotenough/bind.test/inventory/group_vars/all/domain.yml*, where *bind* must be replaced by the name of the service. It is handy for debugging (i.e. browsing the web interface of a service, ssh to a machine that failed to run properly, etc.)

### 5.5.6 Upgrade testing

To verify that a service (*icinga* for instance) can be upgraded from a given Enough version (*2.0.7* for instance), use:

```
$ export ENOUGH_API_TOKEN=XXXXXXX
$ tests/run-upgrade-tests.sh 2.0.7 icinga
...
```

*run-upgrade-tests.sh* performs the following steps:

- checkout the `2.0.7` tag into `../infrastructure-versions/2.0.7/infrastructure`
- run `tox -e icinga` from the `2.0.7` directory and keep the hosts
- run `tox -e icinga` from the current version, re-using the hosts with the icinga version installed from `2.0.7`

### 5.5.7 ssh to a host under test

If *tests/run-tests.sh tox -e chat* was run and the hosts have not been destroyed because the *–enough-no-destroy* option was set, the following can be used to ssh on a host:

```
$ tests/run-tests.sh tests/ssh chat bind-host
debian@bind-host:~$
$ tests/run-tests.sh tests/ssh chat bind-host hostname
bind-host
```

### 5.5.8 Debugging tests

To run the tests manually within the test container:

```
$ tests/run-tests.sh bash
user@6642e3759c43:~/infrastructure$ tox -e flake8
```

Use the `--log-cli-level` switch in order to:

- enable log display during test run (live logging)
- control the test log level

For example:

```
$ tests/run-tests.sh tox -e py3 -- --log-cli-level=INFO -s -x tests/enough/common/
↪test_openstack.py
```

–log-cli-level and following switches are from pytest.

To execute only one test:

- `tests/run-tests.sh tox -e py3 -- tests/enough/common/test_openstack.py::test_heat_definition`

There should not be any leftover after a test involving OpenStack fails, because the fixtures are supposed to thoroughly cleanup. But bugs are to be expected in a test environment and it may be necessary to manually remove leftovers, using the `openstack` command like so:

- `tests/run-tests.sh env OS_CLIENT_CONFIG_FILE=tests/clouds.yml openstack --os-cloud production stack list`
- `tests/run-tests.sh env OS_CLIENT_CONFIG_FILE=tests/clouds.yml openstack --os-cloud clone stack list`

In case leftover are manually deleted using `stack delete` command, the following directory must be manually removed: `.tox/<test environment>/.pytest_cache/`, for example `.tox/py3/.pytest_cache/`.

### 5.5.9 Execute Ansible on the test infrastructure

Display content of `/path/to/a/file` from `bind-host` when `icinga` test environment is used:

```
$ tests/run-tests.sh .tox/icinga/bin/ansible bind-host \
   -i .tox/icinga/.pytest_cache/d/dotenough/icinga.test/inventory \
   -mraw cat /path/to/a/file
```

Check the value of an ansible variable:

```
$ tests/run-tests.sh .tox/icinga/bin/ansible bind-host \
   -i .tox/icinga/.pytest_cache/d/dotenough/icinga.test/inventory \
   -m debug -avar=ansible_host
```

### 5.5.10 Repository layout

The ansible part of the repository groups playbooks and roles in separate directories to reduce the number of files to consider when working on improving a playbook or a service.

- `playbooks/authorized_keys`: distribute SSH public keys
- `playbooks/backup`: daily VMs snapshots
- `playbooks/bind`: DNS server and client
- `playbooks/icinga`: resources monitoring
- `playbooks/infrastructure`: VMs creation and firewalling
- `playbooks/postfix`: outgoing mail relay for all VMs
- etc.

The other scenarii found in the *playbooks* directory are services such as weblate or discourse.

The toplevel directory contains the playbook that applies to the enough.community production environment. It imports playbooks found in the *playbooks* directory.

### 5.5.11 Managing python dependencies

- adding a new dependency: *pipenv install thepackage*
- creating the requirements*.txt files needed to create a distribution: *pipenv run pipenv_to_requirements -f*

## 5.6 Release management

- Prepare the ../release-notes.rst
- Prepare a new version

```
$ version=1.3.0
$ rm -f inventory/hosts.yml inventory/group_vars/all/clouds.yml
$ perl -pi -e "s/^version.*/version = $version/" setup.cfg
$ for i in 1 2 ; do
      python setup.py sdist
      amend=$(git log -1 --oneline | grep --quiet "version $version" && echo --amend)
      git commit $amend -m "version $version" ChangeLog setup.cfg
      git tag -a -f -m "version $version" $version
  done
```

- Build the release locally

```
$ docker rmi enoughcommunity/enough
$ python -m enough.internal.cmd build image
$ docker tag enough:$version enoughcommunity/enough:latest
$ docker tag enough:$version enoughcommunity/enough:$version
$ docker rmi enough:$version
```

- (Optional) manual test the release

```
$ eval "$(docker run --rm enoughcommunity/enough:latest install)"
$ enough --version
```

- Publish the release

```
$ git push ; git push --tags
$ twine upload -s --username enough --password "$ENOUGH_PYPI_PASSWORD" dist/enough-
↪$version.tar.gz
$ docker login --username enoughh --password "$ENOUGH_DOCKER_HUB_PASSWORD"
$ docker push enoughcommunity/enough:latest
$ docker push enoughcommunity/enough:$version
```

- PyPI maintenance
  - if the project does not yet exist

    ```
    $ python setup.py register
    ```

  - trim old versions at https://pypi.python.org/pypi/enough

## 5.7 Team

The *enough.community* maintainers team is always looking for new members willing to help on a regular basis. If you feel like it, *submit a merge request*. After it is merged you will be given credentials.

### 5.7.1 Contact

The contact mail alias is forwarded to:

- Loïc Dachary
- François Poulain

### 5.7.2 Funding

*Funding*

- Loïc Dachary

### 5.7.3 Infrastructure

*DNS*, *Hosting and infrastructure*

- François Poulain
- Loïc Dachary
- Pierre-Louis Bonicoli

### 5.7.4 Weblate

- Loïc Dachary

### 5.7.5 Mattermost

- François Poulain
- Loïc Dachary
- Pierre-Louis Bonicoli

### 5.7.6 Readthedocs

The readthedocs user is *enough-community*.

- Loïc Dachary

## 5.8 Funding

See the funding category of the forum.