Enough Documentation

Release 0.0.0

Enough community

Feb 19, 2020

enough.community

The enough.community infrastructure supports the development of Enough. It is maintained by a *community of individuals* organized horizontally. Anyone is welcome to contribute: learn more in the *contribution guide*.

All resources are deployed and maintained via *ansible*.

CHAPTER 1

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.

1.1 Contact

The contact mail alias is forwarded to:

- Loïc Dachary
- François Poulain
- Louis Vanhaelewyn

1.2 Funding

Funding

- Loïc Dachary

1.3 Infrastructure

DNS, *Hosting and infrastructure*

- François Poulain
- Loïc Dachary
- Pierre-Louis Bonicoli

1.4 Weblate

- Loïc Dachary

1.5 Mattermost

- François Poulain
- Loïc Dachary
- Louis Vanhaelewyn
- Pierre-Louis Bonicoli

1.6 Readthedocs

The readthedocs user is *enough-community*.

- Loïc Dachary

CHAPTER 2

Contribute

This is the contribution guide to the *enough.community* infrastructure which is based on Ansible. 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.

Note: If you want to contribute to the Enoug code base, take a look at the repository.

2.1 Resources

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

2.2 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.

2.3 Organization

All contributors are organized horizontally

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

2.4 Getting started

- `git submodule update --init`
- `apt install virtualenv`
- `deactivate || true ; source bootstrap`
- get OpenStack credentials (ask *anyone in the*) and store then in *openrc.sh*
- `source openrc.sh`
- `openstack server list`: should successfully return nothing on a new tenant
- `cp clouds.yml.example inventories/common/group_vars/all/clouds.yml` and edit to match *openrc.sh*
- `molecule converge -s bind`: create VMs for the scenario *bind* and run ansible playbook defined for this scenario
- `molecule verify -s bind`: run scenario's tests
- `molecule login -s bind --host bind-host`: should ssh to the machine
- `molecule destroy -s bind`: destroy the virtual machine and cleanup the tenant

2.5 Ansible repository layout

The ansible 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.

- `molecule/authorized_keys`: distribute SSH public keys
- `molecule/backup`: daily VMs snapshots
- `molecule/bind`: DNS server and client
- `molecule/letsencrypt-nginx`: nginx reverse proxy with letsencrypt integration
- `molecule/icinga`: resources monitoring
- `molecule/infrastructure`: VMs creation and firewalling
- `molecule/postfix`: outgoing mail relay for all VMs
- `molecule/preprod`: full preproduction environment. See *Integration testing*.
- etc.

The other scenarii found in the *molecule* 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 *molecule* directory.

2.6 Integration testing

Unit tests are welcome, integration tests are mandatory. When modifying a role or a playbook in the directory *molecule/ABC* one is expected to add a test for the new behavior and verify it runs successfully:

- `molecule test -s ABC`

Ansible being declarative for the most part, unit tests are only beneficial to verify loops and conditionals work as expected. For instance by checking a file is created only if **–tag something** is provided. An integration test is necessary to checks if the service is actually doing anything useful. For instance the integration tests for weblate request that the weblate server sends a mail and verify it is relayed by the postfix server.

When possible 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.

After all tests pass, integration with online services must be verified manually inside the preproduction environment.

The value of `ENOUGH_API_TOKEN` below is displayed to signed-in users at https://api.enough.community. Members of the group enough can sign-in, others can request access.

- `ENOUGH_API_TOKEN=XXXXXXX molecule create -s preprod`
- `molecule converge -s preprod`
- at end of converge you will get advertised about the testing subdomain:

```
TASK [debug] *******************************************************************
    ok: [localhost] => {
    "domain": "ndi1nze0mdqk.test.enough.community"
}
```

- `molecule verify -s preprod`
- manually verify *weblate.ndi1nze0mdqk.test.enough.community*, *icinga.ndi1nze0mdqk.test.enough.community*, etc. and integration with online services such as GitHub authentication.
- `molecule destroy -s preprod`

CHAPTER 3

Funding

See the funding category of the forum.

CHAPTER 4

Introduction

The enough CLI controls the infrastructure.

CHAPTER 5

Installation

- Install Docker http://docs.docker.com/engine/installation/
- Copy the following to `~/.bashrc`:

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

- Verify that it works:

```
enough --help
```

CHAPTER 6

Release management

- Prepare a new version
- version=1.3.0 ; 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” $version ; done
- Publish a new version
- twine upload –username enough –password “$ENOUGH_PYPI_PASSWORD” dist/enough-$version.tar.gz
- git push ; git push –tags
- docker rmi enoughcommunity/enough
- python -m enough.internal.cmd build image
- docker tag enough:$version enoughcommunity/enough:$version
- docker tag enough:$version enoughcommunity/enough:latest
- docker login –username enoughh –password “$ENOUGH_DOCKER_HUB_PASSWORD”
- docker push enoughcommunity/enough:latest
- docker push enoughcommunity/enough:$version
- pypi maintenance
- python setup.py register # if the project does not yet exist
- trim old versions at https://pypi.python.org/pypi/enough

CHAPTER 7

Ansible

7.1 Secrets

The default credentials (for Weblate, Discourse etc.) are only suitable for integration testing and must be overriden before deploying on publicly available hosts. The recommended way of doing this is to:

- create a repository in *~/.enough/enough.community/inventory*
- for each files containing secrets i.e. {host,group}_vars/**/*.yml‘) create a matching file in *~/.enough/enough.community/inventory*
- encrypt those files with ansible vault
- share the password to decrypt the files with trusted administrators
- push in a private repository

The encrypted secrets are kept in a private repository to not be publicly exposed to brute force attacks.

7.2 Creation

Manually create *~/.enough/enough.community/inventory/group_vars/all/clouds.yml* by copying *clouds.yml.example* and getting values from *~/openrc.sh* and check it works:

```
$ export OS_CLIENT_CONFIG_FILE=~/.enough/enough.community/group_vars/all/clouds.yml
$ openstack --os-cloud ovh server list
```

```
$ echo domain: enough.community | sudo tee ~/.enough/enough.community/group_vars/all/
↪domain.yml
```

7.2.1 Getting the production repository

```
$ git clone git@lab.enough.community:production/enough.git ~/.enough/enough.community
$ ansible-vault decrypt \
                --vault-password-file ~/.enough/enough.community/vault_pass.txt \
                ~/.enough/enough.community/infrastructure_key
```

7.3 Running

7.3.1 Creating new hosts

Note: do not run the following from a git checkout. If run from sources, the test environment will be used instead of the production environment.

```
$ python -m enough.internal.cmd --domain enough.community host create some-host
$ python -m enough.internal.cmd --domain enough.community host inventory
```

It will set the IP address of the new host into *~/.enough/enough.community/hosts.yml*.

```
all:
  hosts:
    bind-host: {ansible_host: 51.68.89.70}
    wereport-host: {ansible_host: 51.68.88.149}
```

7.3.2 Updating

The ansible repository is run as follows:

```
$ export MOLECULE_FILE=$(pwd)/molecule/preprod/molecule.yml
$ ansible-playbook --private-key ~/.enough/enough.community/infrastructure_key \
                   --vault-password-file=~/.enough/enough.community/vault_pass.txt \
                   -i inventories/common \
                   -i ~/.enough/enough.community/inventory \
                   enough-community-playbook.yml
```

Some hosts contain private information that belong to users who only trust some administrators of the infrastructure. These hosts only have the ssh public keys of the trusted administrators and are listed in a dedicated inventory subdirectory. For instance, the administrator *dachary* owns the the inventory directory *inventories/dachary*. This administrator can then run the playbook on all the common infrastructure as well as all the hosts that can only be accessed by them as follows:

```
ansible-playbook --private-key ~/.enough/enough.community/infrastructure_key \
                 --vault-password-file=~/.enough/enough.community/vault_pass.txt \
                 -i inventories/common \
                 -i inventories/dachary \
                 -i ~/.enough/enough.community/inventory \
                 enough-community-playbook.yml
```

CHAPTER 8

Documentation

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

CHAPTER 9

Hosting and infrastructure

9.1 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 **GRA5**: used for testing by Loïc Dachary
  - Region **SBG5**: used for testing by François Poulain
- **OVH Project: CI**
  - Region **DE1**: GitLab runner
- **OVH Project: Production**
  - Region **GRA5**: all Ansible maintained production VMs

The OVH acount **cl283532-ovh** is dedicated to hosting Enough instances dedicated to an organization or an individual.

- Login as a customer
- OpenStack OVH management

9.2 Security groups

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

9.3 VM naming conventions

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

9.4 Global account name

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

CHAPTER 10

Extending *enough.community*

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

10.1 Overview

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

The development, tuning and corrections of the playbook is done in a separate environment with automated testing. Molecule is the test infrastructure.

Each playbook (for instance *molecule/weblate/weblate-playbook.yml*) is related to a molecule *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 *molecule/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.

10.2 Adding a host

Ansible hosts are defined in *inventories/common/01-hosts.yml*. This file is autogenerated by molecule scenarios and is not meant to be manually modified.

10.3 Adding a scenario

Let's start with a *dummy* scenario.

10.3.1 Copying from an existing scenario

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

```
cp -a molecule/website molecule/dummy
```

10.3.2 Scenario definition

Next, edit *molecule/dummy/molecule.yml*. You could at least:

- edit the scenario's name

```
scenario:
  name: dummy
```

- control the hosts to be created

```
- name: dummy-host
  flavor: "s1-2"
- name: external-host
  flavor: "s1-2"
```

The flavor refers to OVH OpenStack provisionned ressources.

You should then be able to:

- create instances

```
molecule create -s dummy
```

- log into instances

```
molecule login -s dummy --host dummy-host
molecule login -s dummy --host external-host
```

- destroy instances

```
molecule destroy -s dummy
```

About hostnames

For the most part we do not use groups and hostnames matche the service they provide. We use *service*-host as a naming convention. Each given host (e.g. *postfix-host*) matches the corresponding service (e.g. "infrastructure's mail relay").

- if you need to choose a hostname, derive it from the service,
- by using a name for which there already is a scenario (*postfix-host* for instance), you will deploy the corresponding service.

10.3.3 Adding playbooks

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

- others scenarios playbooks, like *molecule/icinga/icinga-playbook.yml* or *molecule/postfix/postfix-playbook.yml*
- the playbook specific to this scenario, here *molecule/icinga/dummy-playbook.yml*, which is intended to be included in *enough-community-playbook.yml*. This playbook may include other playbooks.
- tests specific playbooks, starting with *test*, e.g. *molecule/icinga/test-dummy-playbook.yml*.

Once the playbooks are added, you should be able to check their syntax and run them with:

```
molecule syntax -s dummy
molecule converge -s dummy
```

10.3.4 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.

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

The test can be launched with

```
molecule verify -s dummy
```

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.

You can launch a *destroy, create, converge, verify, destroy* cycle with

```
molecule test -s dummy
```

10.3.5 Interaction with others scenarios

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

You will also be interested by:

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

10.3.6 Documentation

You are kindly invited to document your scenario in *docs*. Most playbooks are documented in a dedicated file included from *docs/index.rst*.

10.3.7 Tweaking hosts

You can set ssh port, choose OS image and set default user by tweaking *hosts-base.yml*.

CHAPTER 11

DNS

11.1 Registrar

The *enough.community* domain name is registered at Gandi under the user EC8591-GANDI.

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.

```
ansible-playbook -l bind-host \
                 --private-key infrastructure_key \
                 -i inventories/common \
                 enough-community-playbook.yml
```

11.2 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.

11.3 Zones

11.3.1 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
- Manually maintained records are added to the bind playbook.
- The *SPF* **TXT** record help *send mail* successfully.

11.3.2 test.enough.community

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

It can be updated locally by the *debian* user via `nsupdate`. This enables any enough.community's administrator to setup new preproduction testing subdomains. Exemple:

```
- 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
```

11.4 VMs resolvers

Each VM is set to use *ns1.enough.community* as a resolver via the bind-client playbook which also sets the FQDN.

CHAPTER 12

Intrusion Detection System

12.1 Wazuh

The Wazuh server/manager is installed on a dedicated host and all other hosts run an agent. The roles used by the wazuh playbook are from a submodule including a short lived fork of the wazuh-ansible repository. All commits unique to the fork must match a pull request so they are eventually merged.

12.2 Notifications

All notifications are sent to *ids@enough.community*.

CHAPTER 13

SSH public keys

Each *team* member with access to the infrastructure should have their ssh public key in the ssh_keys directory so it is added to the *~debian/.ssh/authorized_keys* file.

CHAPTER 14

Backups and recovery

14.1 Backup policy

Each VM is snapshoted daily and snapshots older than 30 days are removed.

14.2 Disaster recovery

The VMs are cheap and do not provide any kind of guarantee: all data they contain can be lost. To recover a lost production VM:

- login debian@ansible.enough.community and get OpenStack credentials from *~/openrc.sh* or *ask a team member*.
- cd /srv/checkout

14.2.1 If the virtual machine is cattle

- Delete the broken machine if it is still around, e.g. `openstack stack delete website-host`
- Create the machine again as if it was new
- *Run ansible* so the DNS updates with the IP of the newly created VM

14.2.2 If the virtual machine is a pet

- Get the name of the latest backup with `openstack image list --private`
- Rename the broken machine if it is still around, e.g. `openstack server set --name packages-destroyed packages-host`
- Get the flavor for the machine from `molecule/preprod/molecule.yml`

- Create a new machine from the backup, e.g. `openstack server create --flavor s1-2 --image 2018-05-14-packages-host --security-group infrastructure --wait packages-host`
- Edit `inventories/common/01-hosts.yml` and replace the IP of the broken machine with the IP of the new machine
- Clear the ansible cache `rm -fr ~/.ansible`
- *Run ansible* so the DNS updates with the IP of the newly created VM
- Reboot the machine, in case it had IPs from before the DNS was updated by ansible
- *Run ansible* so the DNS updates with the IP of the newly created VM

14.3 Disaster recover exercize

14.3.1 If the virtual machine is cattle

- Remove the machine, e.g. `openstack server delete website-host`

14.3.2 If the virtual machine is a pet

- Rename the machine, e.g. `openstack server set --name packages-destroyed packages-host`
- Suspend, e.g. `openstack server suspend packages-destroyed`
- Remove the machine when the recovery is successfull, e.g. `openstack server remove packages-destroyed`

CHAPTER 15

Postfix mail server

Each VM installed via Ansible is able to send emails from the *enough.community* domain.

15.1 Postfix mail relay

A mail relay based on Postfix is defined on the host *postfix-host*, via the playbook *molecule/postfix/postfix-relay-playbook.yml*, based on the Postfix DebOps role.

It is configured as an open relay using *smtps*. The relaying restrictions are set in *firewall* using OpenStack.

15.2 Postfix mail satellite

A Postfix satellite is defined on each host (except for *postfix-host*), via the playbook *molecule/postfix/postfix-client-playbook.yml*, based on the Postfix DebOps role.

CHAPTER 16

Monitoring architecture

16.1 Icinga2 overview

Icinga2 is very flexible and doesn't impose monitoring architecture. So we have to define it. The simplest is to follow the master with clients setup. It implies:

- getting a master server (including web GUI),
- getting a client for each VM on which you would execute checks.

Icinga2 uses the same software for clients and masters and their configuration defines if one is master or client. We can use the same configuration objects for master and clients.

Our master server deployment is defined in *molecule/icinga/roles/icinga2*.

Our client deployment is defined in *molecule/icinga/roles/icinga2_client*.

Executables checks are managed locally on each computer, as well as required sudo permissions. To deploy them, a common role is defined in *molecule/icinga/roles/icinga2_common*.

Icinga2 define "zones", which is a way to control information sharing over the monitoring infrastructure. We define:

- a global zone for the configuration shared among all the cluster;
- a master zone for the master;
- a client zone for each client which the master zone as a parent.

Client dont't know about each other (the master will distribute only what is required).

16.2 Molecule test environment

In *molecule/icinga*, we define a test environment with a master and a client.

A set up for Icinga2 needs master and clients, NginX, Icingaweb2 and Let's Encrypt. Tests are checking that the playbook is well executed.

16.3 Apply logic style

Icinga2 uses the "apply logic style". All behavior is described using a language (including list and associative array), as an host attribute (e.g. a list of hardware block devices, a list of mounted volumes, a list of vhosts and some associated attributes, a list of process you would like to check and their associated limits, a list of git repos to be checked, etc.)

Based on those attributes provided, generic service can be defined. Here is how one can check all the certificates of all the vhosts which are declared to use TLS:

```
apply Service ""Check TLS certificate "" for (http_vhost => config in host.vars.http_
↪vhosts) {
    import ""generic-service""

    check_command = ""http""
    vars.http_address = config.http_vhost
    command_endpoint = "" ... ""
    vars.http_certificate = 21
    vars.http_sni = true

    vars += config
    assign where config.http_ssl == true
}
```

with n host which contains this declaration:

```
vars.http_vhosts[""Forum""] = {
  http_vhost = ""forum.enough.community""
  http_ssl = true
}
```

The main monitoring configuration for enough.community is available in *molecule/icinga/roles/icinga2/files/services/* and deployed in the global Icinga zone, thus available to all the cluster.

There are checks for vhosts, DNS zones consistency, DNS views consistency, attended processes, attended vhosts, attended output IPs, git repos, mails queues, services banners (ssh, smtp, etc.), upgrades, running kernels, mailname consistency, volumes, databases, etc.

CHAPTER 17

Monitoring howto

Icinga2 follows an "apply logic style" for its configuration but it does not disallow adding isolated services, to handle exceptions to the rules.

Most of the service definitions are based on predefined commands which are documented here.

17.1 Monitoring deployment

Monitoring is deployed by importing the *molecule/icinga/icinga-playbook.yml* playbook. The Icinga2 master is *icinga-host*. See also *inventories/common/host_vars/icinga-host/monitoring.yml* for specific deployment attributes: icingaweb credentials, https, virtualhost fqdn.

Each host is monitored by default.

To disable monitoring for some host, you have to define a host variable `not_monitored`.

17.1.1 Base system monitoring

For each host we:

- check ping (default host check in Icinga)
- check ssh
- check apt
- check etckeeper
- check icinga
- check load
- check procs
- check swap when `vars.swap` is defined
- check users

- check run_kernel (check if it run the most up-to-date kernel)
- check fail2ban process
- check sshd process
- check rsyslogd process
- check icinga2 process
- check cron process

17.1.2 Git repos monitoring

A host can declare a git repo to be checked (designed originally for *etckeeper*):

```
vars.repos["Bling"] = {
  dir = "/var/git/bling"
}
```

The git check command is sudoed.

Example of use in a role: *molecule/icinga/roles/deploy_dummy_monitoring_objects/tasks/main.yml*.

17.1.3 Disk and partitions monitoring

A host can declare any partition to be checked:

```
vars.disks["disk"] = {
}
vars.disks["disk /"] = {
  disk_partitions = "/"
}
vars.disks["disk /var"] = {
  disk_partitions = "/var"
}
vars.disks["disk /tmp"] = {
  disk_partitions = "/tmp"
}
```

17.1.4 Processes monitoring

A host can declare any process presence to be checked:

```
vars.process["Incron"] = {
  procs_command = "incrond"
  procs_critical = "1:1"
}
```

Example of use in a role: *molecule/icinga/roles/deploy_dummy_monitoring_objects/tasks/main.yml*.

17.1.5 Mail sending monitoring

A host can declare any non-null value in `vars.sendmail`. Then mailname, mail queue and process are checked.

It's currently not sufficient.

I wrote some time ago a qshape based test which is suitable to detect delivery problems for mass mailling (much better than the mail queue which can legitimately grows when needed). But it is not adapted for sparse emailling.

So, some nice projects would be of interest to monitor our ability to send emails:

- check qshape.
- check rbls.
- a mail loop test (verify the self-delivery of a sent mail gone via another relay)
- a mail delivery test (verify the delivery of a sent mail in some of the majors mails domains)

17.1.6 Web services monitoring

A host can declare hosting web at a given fqdn:

```
vars.http_vhosts["Secure Drop Forum"] = {
  http_vhost = "forum.enough.community"
  http_uri = "/c/devops"
  http_ssl = true
  http_string = "devops discussions"
}
```

- Each fqdn is processed via `check_http` from Icinga master and should provide `http_string` in answer's body.
- Each fqdn is processed via `check_http` from Icinga master and should *not* provide some strings in the answer. It is useful to prevent from accidentally deploy spywares. For now, spywares checked are:
  - googleapis.com
  - cloudflare.com
  - google-analytics.com
  - gravatar.com
- If `http_ssl = true` the check is processes using https and the TLS certificate is retrieved for validity check.

Moreover if a host declare `vars.httpd = "apache"` or `vars.httpd = "apache2"` or `vars.httpd = "nginx"`, then processes check are executed.

If a host declare `vars.sqlserver = "mysql"` or `vars.sqlserver = "mariadb"` or `vars.sqlserver = "pgsql"`, then processes check are executed.

It is probably easily feasible to associate a list of scripts to each fqdn for more advanced checks (check result of a POST, etc.) if needed.

Example of use in a role: *molecule/weblate/roles/weblate/tasks/monitoring.yml*.

Since monitoring *http vhosts* happens often in *enough.community*, an Ansible role helps to declare it:

```
- role: monitor_http_vhost
  http_vhost_name: Secure Drop Forum
  http_vhost_fqdn: "forum.{{ domain }}"
  http_vhost_uri: /c/devops
  http_vhost_string: "devops discussions"
  http_vhost_https: true
```

17.1.7 Torified Web services monitoring

Similarly to *http_vhosts*, a host can declare a *tor_http_vhosts* dictionnary. The main difference is that it is not a *fqdn* which is transmitted, but the path of the service hostname. An Ansible role helps to declare it:

```
- role: monitor_tor_http_vhost
  tor_hostname_file: /var/lib/tor/services/cloud/hostname
  tor_http_vhost_name: Cloud
  tor_http_vhost_uri: "/login"
  tor_http_vhost_string: "Forgot password"
```

Note: For now the only handled case concerns plain http over tor. TLS hasn't yet been defined.

17.1.8 DNS service monitoring

A host can declare hosted zones files which can be checked via `named-checkzone` (syntax consistency) and `check_whois` (domain expiration):

```
/* Define zones and files for checks */
vars.zones["Secure Drop Club"] = {
  fqdn = "enough.community"
  file = "/etc/bind/zones/masters/enough.community"
  view = "external"
}
```

Example of use in a role: *molecule/bind/roles/monitoring-bind/tasks/main.yml*.

Maybe we could add a check dig on the A and NS records, and eventually use `zonemaster` or a webservice providing `zonemaster` results.

17.2 Monitoring tweaking

17.2.1 Service templates

A host can set a prefered service template, using the icinga variable `vars.service_template`.

The templates can be found in *molecule/icinga/roles/icinga2/files/templates.conf*.

17.2.2 Hosts vars

A host can define a list of lines to be added to its icinga configuration, using the Ansible variable `monitoring_host_vars`. Se e.g. `inventories/common/host_vars/icinga-host/monitoring.yml` for an example.

Default is empty.

CHAPTER 18

Weblate

weblate.enough.community is installed with docker.

See also *host_vars/weblate-host/weblate.yml*.

The docker-compose file is adapted from the one found in the weblate repository

CHAPTER 19

GitLab

lab.enough.community is installed with docker.

The configuration variables are set in *inventories/common/host_vars/gitlab-host/gitlab.yml* at the root of the repository. It can be copied from *molecule/gitlab/roles/gitlab/defaults/main.yml*.

- *gitlab_password*: database password
- *gitlab_shared_runners_registration_token*: runner registration token predefined for tests
- *gitlab_secrets_db_key_base*, *gitlab_secrets_otp_key_base* and *gitlab_secrets_secret_key_base*: unique keys that can be generated with *pwgen -Bsv1 64*
- *gitlab_os_**: default to the OpenStack tenant variables. In production they should be set to a dedicated tenant, entirely separated from production, because it will be used by all commits pushed to the repository.

CHAPTER 20

Mattermost

chat.enough.community is installed *with docker <https://docs.mattermost.com/install/prod-docker.html>*. The configuration is done via the admin console web interface.

Using the CLI:

```
cd /srv/mattermost
docker-compose -f docker-compose-infrastructure.yml exec app platform
```

Entering the Mattermost container:

```
cd /srv/mattermost
docker-compose -f docker-compose-infrastructure.yml exec app sh
```

CHAPTER 21

Enough

cloud.enough.community is installed with docker.

The `/var/lib/docker` directory is mounted on a 3 replica volume and should be manually backup from time to time to keep the history. If there is not enough space, it can be resized with:

```
$ openstack volume set --size 200 cloud-volume
$ ssh debian@cloud.enough.community
$ sudo resize2fs /dev/sdb
```

Note that the `size` in the ansible role for the `os_volume` tasks is only used when the volume is created and cannot be used to shrink or enlarge the volume.