Project title: Community Networks Testbed for the Future Internet

Management guide for deployed testbed (Year 4)

Deliverable number: D3.4

Version 1.2

This project has received funding from the European Union's Seventh Programme for research, technological development and demonstration under grant agreement number 288535.

Project acronym: CONFINE
Project full title: Community Networks Testbed for the Future Internet
Type of contract: Integrated project (IP)
Contract number: 288535
Project URL: http://confine-project.eu

Editor:	Ivan Vilata (Pangea)
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual delivery date:	September 30, 2015
Actual delivery date:	September 15, 2015
Suggested readers:	Project partners
Number of pages:	38
Keywords:	WP3, documentation, guide, manual, community networks, testbed, deployment, operation, management, administration, controller
Authors:	Santiago Lamora, Ivan Vilata (Pangea)
Peer review:	Pau Escrich (Guifi), Jorge Florit (UPC), Ester López (UPC), Claudio Pisa (Ninux)

Abstract

This report introduces the testbed administrator's guide which has been developed during the fourth year of the CONFINE Project. The guide is briefly described here, and a whole copy of it is included as an appendix. A short summary of the current deployment status of the Community-Lab testbed is also presented.

Contents

1. Introduction

This chapter presents a justification of the contents of the deliverable and how it is related with other ones.

1.1. Contents of this deliverable

The *Description of work* document for CONFINE mentions this deliverable as instrumental to reflect progress of Objective 2:

> **Objective 2. Provision of an experimental facility** to the research community to support experimentally-driven research on these community networks:
>
> - Operation of the testbed, including management of the testbed (T3.1 and D3.1, D3.2, D3.4), support services for research users (T3.2 and D3.1, D3.3), and deployment of new nodes and links specific for the testbed (T3.3 and D3.1, **D3.4**).

As well as Tasks 3.1 and 3.3:

> **T3.1 Experimental facility provision (management of the testbed)** (from M06 to M48): This task is responsible for the provision of a continuously operating testbed and the provision of the specified tools, services and features to allow users carrying out experimental research. This task includes the administration of the testbed, monitoring of services and traffic.
>
> The first results of task T3.1 will be reported in D3.1 (M12), which will describe the initial plan for operation of the experimental facility based on the integration of the different results of WP2 and will also include the license and conditions of usage for experimenters and the structure for the governance of the testbed. The following deliverables D3.2 (M24), and **D3.4** (M36), be testbed management guides which include the description of the extension and integration of new features in the experimental facility made during in each reporting period.
>
> **T3.3 Deployment of new nodes** (from M7 to M36): In this task the physical deployment of new nodes in the testbed will be undertaken. The new nodes deployed in the first year will use the customized system software developed in Task 2.2. Feedback from user and usage experiments in the second year will broaden the required hardware platforms to be used in the nodes. New sets of nodes with different features will therefore be deployed and integrated over the duration of the project. From year two on the updates and particular system software requirements will be provided by Task 2.5.
>
> The results of Task 3.3 will be reported in D3.1 (M12) containing an initial deployment guide for extending the testbed with new nodes. The deliverables **D3.4** (M36) will report on the nodes in the testbed in the first three years.

Please note that D3.4 incorrectly appears as scheduled for M36 in the general list of deliverables and the task descriptions for WP3 in pages 32, 41 and 42 of the DoW, but it is correctly scheduled for M48 in the list of deliverables for WP3 in page 42 of the DoW.

Thus, the main contents of this report are:

- An introduction to the testbed deployment and management documentation of CONFINE, which has been collected and structured during the fourth year of the Project in the form of an administrator's guide. This includes a description of the guide and the purpose of each of its parts.
- The aforementioned guide, which is included as an appendix.
- A short summary of the status of the deployment of the controller and nodes of the Community-Lab testbed as of September 2015.

1.2. Relationship to other CONFINE deliverables

The present deliverable D3.4 is related with the following CONFINE deliverables:

D2.1 Initial system software and services of the testbed (M12)

D2.3 System enhancements (Year 2) (M24)

D2.5 System enhancements (Year 3) (M36)

D2.7 Software system for the testbed (M48)

D3.4 covers the user documentation of the software and testbed system developed by the CONFINE project and described in deliverables D2.1, D2.3, D2.5 and D2.7. Any mentioned recent developments are described in D2.7. The documentation is considered an integral part of the final software package described in D2.7.

D3.1 Operation and support guides of the testbed (M12)

D3.4 documents the procedures needed to setup and maintain CONFINE testbed deployments such as the Community-Lab testbed as introduced in chapter 3 (*Initial deployment guide and deployment of CONFINE nodes*) of deliverable 3.1.

D3.2 Initial management guide of the testbed (M24)

D3.4 describes the configuration of the testbed management features of Controller software described in section 2.2 (*Controller and management tools for the testbed*) of deliverable 3.2.

D3.3 User and deployment guides (Year 3) (M36)

D3.4 complements the *Community-Lab/CONFINE user's guide* included in appendix A of deliverable 3.3 so as to cover the rest of user profiles related with CONFINE testbeds, i.e. those of management and operation, besides simple testbed usage.

2. Administrator's guide

In the fourth year of the CONFINE project (Oct 2014–Sep 2015) (Y4), a considerable effort has been put in providing stakeholders of the CONFINE Project with the knowledge needed to maintain CONFINE testbeds on their own. The main reason for this effort is the importance of releasing this knowledge for the long-term survival and adoption of the Community-Lab testbed[1] in particular and CONFINE software in general once the Project no longer exists formally.

The **CONFINE administrators's guide**[2] (see appendix A) is the result of collecting and structuring existing documentation and practices on the administration of Community-Lab and other CONFINE testbeds during the life of the Project. The guide covers the most frequent tasks and issues that someone may face as an operator when creating a brand new CONFINE testbed or administering an existing one. In the same vein as the *CONFINE user's guide*[3] and as a complement of it, the administrator's guide can either be used as reference documentation to lookup while deploying or maintaining a CONFINE testbed (or Community-Lab itself), or it can be read from beginning to end as a manual. It also uses a lean style to reflect the sequences of steps that the operator may take in the most common cases.

This guide is part of the CONFINE wiki and it is divided into the following sections, which are mostly self-contained and can be regarded as individual guides:

Introduction Presents the roles of the controller server and Controller software in the CONFINE architecture.

Controller installation Indicates the requirements of a controller server and the steps needed to install and start the Controller software and its dependencies.

Controller configuration Helps the reader configure basic Controller settings to tune its operation and the testbed.

Testbed preparation Covers the configuration of high-level testbed elements, infrastructure and services.

Managing users and groups Provides help on handling user registration and tuning user and group permissions.

Testbed maintenance Gives hints to keep a testbed and its controller server healthy and performant during their lifetime.

Troubleshooting Helps in fixing several known issues that may appear during testbed operation.

3. Testbed deployment

Y4 has seen a **stabilization in deployment growth** by initial Project partners, with a remarkable increase in Greece by the Sarantaporo.gr network (which entered CONFINE in Open Call 2), supported by the Athens Metropolitan Wireless Network (AWMN), which has also added many nodes given the mature state of the Project. From 2014-09-01, a total of 47 new nodes have been registered by different groups as shown in the table below, extracted from the Community-Lab node list[1]:

Group	New nodes
AWMN	13
Guifi.net	5
ninux.org	4
Sarantaporo.gr	18
UPC	4
Wireless België	3
Total	47

Regarding the operation state of the whole testbed, a number of searches on the node list with different filters applied on the nodes' set (expected) state and current (actual) state yield the following numbers:

Set state	Current state	Nodes	%
production	production	111	58.4%
production	offline	61	32.1%
production	other	3	1.6%
safe	safe	3	1.6%
safe	offline	12	6.3%
Total active nodes		190	100.0%

Please note that these values are highly volatile since they are strongly affected by community network connectivity, server load and other factors. The ones above reflect the state of Community-Lab as of 2015-09-04 12:00 CEST. 20 more nodes are actually registered in *failure* set state, which is used for assorted purposes like testing nodes, temporarily or temptatively retiring them, or marking nodes as templates, thus they have not been included in the counts.

All in all, the number of completely **working nodes** (i.e. 111 in production state) represents approximately **one third of the active nodes** in Community-Lab, which is an amount of working nodes similar to that at the end of the third year of the CONFINE project (Oct 2013–Sep 2014) (Y3). This can be explained by the imminent approach of the recently published, stable *Confined* release of CONFINE software, which caused some node administrators to postpone firmware upgrades so as to save upgrade cycles.

[1] https://controller.community-lab.net/admin/nodes/node/?my_nodes=False

With the recent release of the new stable CONFINE software, we are in a phase of **mass upgrade of nodes**, which implies the revision of many nodes that have not been upgraded or checked in a long time. This shall result in much increased numbers of working nodes in the short term (during September and until the end of 2015), rendering a larger, more diverse and stable testbed for researchers to use in the future.

4. Conclusions

With the main objective of ensuring the adoption, maintainability and sustainability of the Community-Lab testbed and the CONFINE software, the need was clear for complete and usable documentation that allows those involved with Community-Lab or interested in CONFINE testbeds to operate them even in the case that the current, live support channels of the CONFINE testbed become missing after the official end of the Project. The *Community-Lab/CONFINE administrator's guide* (included in this deliverable) is the result of the effort to provide the most complete software package and related documentation and knowledge to the current and potential CONFINE Project stakeholders.

Along with the software documentation effort, the Community-Lab testbed is in the process of upgrading its operational infrastructure and nodes to the most stable release of CONFINE software so far, so as to ensure that researchers and community network members can make the most comfortable and profitable use of this innovative experimental platform in the future.

A. Community-Lab/CONFINE administrator's guide

A step-by-step guide to deploying and maintaining a CONFINE testbed

(From https://wiki.confine-project.eu/admin:start as of 2015-09-04.)

A.1. Introduction

This guide describes the tasks needed to **deploy and operate a testbed** based on the software released by the CONFINE Project[4], a European FP7 project intended to develop the tools that allow researchers to deploy, run, monitor and experiment with services, protocols and applications on real-world community networks. These instructions also apply to Community-Lab[1], CONFINE's reference testbed based on several community networks.

The current implementation of CONFINE testbeds relies on a **central controller** which provides a web interface for human interaction with several tools and a set of APIs for programs. It also provides other important services to testbed nodes. For more technical information on the Controller software, see [5].

In terms of the CONFINE architecture[6], the Controller software implements a *testbed server* which provides an entry point to the *Registry REST API*[7] and which acts as a *management network gateway server* (see [8]).

Thus, this guide is centered on the **configuration and maintenance** of a server running the **Controller software**, and other servers run by testbed operators like gateway servers. For the administration of testbed nodes, please refer to Community-Lab/CONFINE user's guide[3].

> *Please note that, throughout the guide and unless explicitly stated otherwise, whenever the reader is instructed to log into the Controller web interface, it is assumed that the used account possesses **superuser privileges**.*
>
> *Also note that command line operations are prefixed with `$` or `#` depending on whether the command must be run as a plain system user (which holds a Controller deployment) or as `root`, respectively.*

A.2. Controller installation

A.2.1. Requirements

Before you set up a CONFINE testbed you need **a complete IPv6 `/48` prefix** to be used exclusively for the testbed (see [8, 9]). You may use a public prefix (even a 6to4[1] would

[1] http://en.wikipedia.org/wiki/6to4

suffice), but the easiest, most durable, resilient, and recommended approach is to generate a new random ULA[2] prefix. For that you may use:

- Web-based generators like SixXS'[3] or Simple DNS Plus'[4], or
- The following Unix command, which should yield an output similar to the one shown below (hexadecimal digits after `fd` shall differ):

```
$ random5B=$(hexdump -n5 -e '/1 "%02x "' /dev/urandom)
$ printf 'fd%s:%s%s:%s%s::/48\n' $random5B; unset random5B
fdc0:7246:b03f::/48
```

You also need an **identifier-like name** for your testbed. For instance, for *My Testbed*, `mytestbed` shall be used.

It is also advisable that you have a **contact email address** for support and other administrative matters. Here we use the sample address `mytestbed@example.com`.

For installing the Controller software it is recommended the use of a **dedicated server**. The Controller needs a running *Debian Wheezy or Jessie* installation, the cleaner the better. It is however highly important that the server is already *able to send email* to arbitrary addresses for notifications (the configuration of the mail server falls outside of the scope of this guide, though). The server should be *reachable by nodes and users* at the IP level; at the very least ports 80/tcp or 443/tcp should be available for the web interface, and ports 655/udp and 655/tcp for the *tinc* overlay supporting the management network.

A **virtual machine or container** should be enough to host the controller server. In the later case, the server should be allowed to use the devices `tun` (for *tinc* interfaces) and `fuse`, and to mount its own filesystems (for node firmware generation).

A.2.2. Deployment

We will be using the `deploy-controller.sh` script included in Controller sources to automate the installation steps. In case of errors, or if the installation stops prematurely, please take note of the first unsuccessful command (in bold letters). Then refer to Installation[5] in order to proceed to step-by-step installation commands starting from the failing one. If you still cannot fix the issue, please look at the Controller documentation[5] or ask for help in the `confine-devel` mailing list[10].

Please note that some errors are inocuous or expected; they will be pointed out below. Also note that the script installs NginX, which is recommended as the web server for the Controller over Apache (which is still currently used by VCT[11]). Thus, web server configuration examples from this point on will be specific for NginX unless noted otherwise.

To begin with the installation of the Controller software after preparing your dedicated server, upload the latest version of deploy-controller.sh[6] to the server. Log into the server (e.g. via

[2] https://en.wikipedia.org/wiki/Unique_local_address
[3] https://www.sixxs.net/tools/grh/ula/
[4] http://www.simpledns.com/private-ipv6.aspx
[5] https://wiki.confine-project.eu/soft:server-installation
[6] https://redmine.confine-project.eu/projects/controller/repository/revisions/master/raw/scripts/deploy-controller.sh

SSH) as `root` and define the following shell variables for the testbed name and management network prefix of your choosing, for example (please use different values!):

```
# TESTBED=mytestbed
# MGMT_PFX=fdc0:7246:b03f::/48
```

Now run the installation script using the Bash shell (standard `sh` will not work):

```
# bash /path/to/deploy-controller.sh -t local -j "$TESTBED" -m "$MGMT_PFX"
```

After downloading and installing software dependencies for the Controller, you will be asked to create an initial **superuser account**. In this example we create a generic account with the testbed contact address, but accounts belonging to normal personal users may act as superusers as well (if granted permission by another superuser). Please take the time to choose and provide proper values to the following questions asked by the installation process (example answers are shown):

```
Username: super  # identifier-like, no spaces or punctuation
Email Address: mytestbed@example.com  # testbed contact/support address
Name: Super User  # free form, spaces and punctuation allowed
Password:   # some secure password
Password (again):   # same secure password
Superuser created successfully.
```

The installation script proceeds to configure the *tinc* daemon for the management network overlay. The script may fail to reload the daemon, but this is expected since *tinc* may not be running yet.

A **certificate for controller APIs** is created next. The installation process will ask for the usual items for certificate request creation, except for the common name, which is preconfigured to the server's address in the management network as shown below (along with example answers):

```
Country Name (2 letter code) []: ES
State or Province Name (full name) []: CT
Locality Name (eg, city) []: Barcelona
Organization Name (eg, company) []: My Organization
Organizational Unit Name (eg, section) []: My Testbed
Email Address []:  mytestbed@example.com  # testbed contact/support address
Common Name: fdc0:7246:b03f::2  # automatically set
writing new certificate to '/home/confine/mytestbed/pki/cert'
```

You may see some `diff` errors about missing files when the script is configuring NginX; this is normal since there is no previous configuration. Stopping `celerybeat` may also fail since it may not be runnnig yet, and Apache restart may fail as well since NginX is being used instead; these errors are inocuous as well.

```
... seems that everything went better than expected :)
```

This line marks the end of the installation. After this point you should be able to access the controller web UI via HTTP and log in with the superuser you just created above. Otherwise, review the output from the installation commands to spot possible errors.

The installation should have created a new **system user** (a Unix account named `confine` by default) and a **Controller deployment directory** for the testbed inside of the user's home directory (e.g. `~confine/mytestbed`). The system user is not related with the Controller superuser account created above, and besides owning the Controller data and configuration files under its home directory, it is meant to manage the Controller by executing the `manage.py` script, which is also part of the deployment.

The system user is not given a password though, so to allow it to log into the controller server a new password should be set (e.g. running `passwd confine`) or SSH authorized keys be added (e.g. to `~confine/.ssh/authorized_keys`). Now you should be able to run a command like `ssh confine@controller.example.com`, where `controller.example.com` is the DNS name of the server where the Controller is installed.

A.2.3. Management address

A final step is needed to ensure the presence of the management network address early on boot. Otherwise, testbed services may start before *tinc* and they would fail to listen on the management address (see Nginx failed to start. Cannot assign requested address?[7] for an example case). Run `ip -6 addr show dev confine` and note down the `ADDRESS/PREFIX` after `inet6`. Edit `/etc/network/interfaces` and add the following lines at the end of the stanza of the main network interface (usually `eth0`):

```
# Ensure that the management address is present early on boot.
up ip addr add ADDRESS dev $IFACE  # leave the "/PREFIX" out
up while [ "$(ip -6 addr show tentative)" ]; do sleep 1; done
```

A.2.4. HTTPS

We recommend that you customize your NginX configuration to serve the web interface over HTTPS instead of HTTP. Just take care not to override the configuration and certificate used to listen on the management network address. Using wildcard addresses on the HTTPS-protected web interface should be fine, though. For an HTTPS-only setup, you may add this `server` entry to `/etc/nginx/conf.d/$TESTBED.conf`:

```
server {
    listen 80;
    listen [::]:80 ipv6only=on;
    return 301 https://$host$request_uri;
}
```

Then in the existing plain HTTP `server` entry, change the `listen` options to:

```
listen 443 ssl;
listen [::]:443 ipv6only=on ssl;
```

Add the `ssl_certificate` and `ssl_certificate_key` options pointing to your own files, and restart NginX with `service nginx restart`.

[7] https://serverfault.com/questions/421460/nginx-failed-to-start-cannot-assign-requested-address

A.3. Controller configuration

Once you have finished the installation of the Controller, you need to provide some first-time configuration items that must be placed in the Controller **settings file** by the time its software services are started.

Log into the controller server (e.g. via SSH) as the system user created for running the Controller software. Then open with your favourite editor the settings file of the Controller. For instance, for the testbed named `mytestbed` (use your own system user and testbed name if different):

```
$ ssh confine@controller.example.com  # in your computer
$ vim ~/mytestbed/mytestbed/settings.py  # in the controller
```

Then update the following settings in order to have a fully functional Controller:

- Customize your site name and URL (they will be used in user registration and notification emails):

  ```
  SITE_URL = 'https://controller.example.com'
  SITE_NAME = 'My Testbed'
  SITE_VERBOSE_NAME = 'My Testbed Management Site'
  ```

- If the local MTA is not able to send messages to arbitrary email addresses, you may want to configure delivery via an external SMTP server:

  ```
  EMAIL_HOST = 'smtp.example.com'
  EMAIL_PORT = 465  # 25 by default
  EMAIL_HOST_USER = 'mytestbed'  # if authentication is needed
  EMAIL_HOST_PASSWORD = 's3cr3t'  # if authentication is needed
  EMAIL_USE_TLS = True  # False by default
  ```

- Configure the source address for normal Controller emails and notifications:

  ```
  DEFAULT_FROM_EMAIL = 'mytestbed@example.com'
  ```

- Configure the recipients and source of administrative emails. The recipients will get code error or malfunction notifications (e.g. when some testbed service breaks or is down), and they should be able to act directly on the Controller installation; superusers use the web interface instead and may not be part of this group. See [12] for more details.

  ```
  ADMINS = (('George', 'george@example.com'),
            ('Mary', 'mary@example.com'))
  SERVER_EMAIL = 'mytestbed@example.com'  # mail address error messages come from
  ```

- Choose the **user registration policy** between 'OPEN', 'RESTRICTED' or 'CLOSED', and the account approval contact address (see User registration [A.5.1] for more information):

  ```
  USERS_REGISTRATION_MODE = 'RESTRICTED'  # 'OPEN' by default
  EMAIL_REGISTRATION_APPROVE = 'mytestbed@example.com'
  ```

- Define which **node architectures** are supported by the testbed and which one is the default (please note that the *Confined* release of Node software only supports 32-bit kernels, see Base images [A.4.4.1] for more information):

  ```
  NODES_NODE_ARCHS = (
      ('i586', 'i586'),  # value and visible title
  ```

```
    ('i686', 'i686'),
)
NODES_NODE_ARCH_DFLT = 'i686'
```

- Configure the default **zoom and coordinates of the map** where the nodes are shown based on their geoposition (if any):

```
GIS_MAP_CENTER = {'lat': 0.0, 'lng': 0.0}
GIS_MAP_ZOOM = 10  # value between 8 (farther) and 15 (nearer)
```

To conclude, just restart the Controller services to **apply changes** on the configuration:

```
$ sudo python ~/mytestbed/manage.py restartservices
```

A.4. Testbed preparation

Before the testbed becomes functional for its users and nodes, some elements must be configured. Here we will see how to use the Controller to setup these features.

A.4.1. Islands

The CONFINE architecture allows to **replicate** or **distribute** certain services to provide convenient properties like **redundancy**, **caching** or **locality**. Especially in the case of community networks, where access to the Internet is not always available, closer services may be more convenient for some components than farther ones, which may have worse connectivity or be completely unreachable.

CONFINE uses the concept of *island* as an attribute of API endpoints, *tinc* addresses, nodes and hosts, that can help choose the most interesting provider of a service among a set of different ones, by conveying an **informal indication of network proximity**. For instance, a node may choose to connect to a *tinc* address of another host which gives access to the management network (i.e. a management network gateway) if that address is in the same island as the node, rather than connecting to an address of another host which is in the Internet or in some undefined island. The same may go for accessing the Registry API, an NTP server, or any random service run by a host known to the testbed registry.

To create an island in your testbed:

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Islands* icon to get the list of islands in the testbed.
3. Click on the *Add island* button to register a new island.
4. Provide an informative name (e.g. *Internet* or *My network campus cloud*) and a description.
5. Click on *Save*.

Registering all network islands beforehand is not necessary, but adding islands for the Internet and other networks your controller is connected to is recommended.

A.4.2. The controller server

Although the Controller installation already creates a server entry for this controller in the testbed registry, the entry still needs some changes.

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Servers* icon to get the list of servers in the testbed. Click on the only existing server.
3. Set informative name and description values for the server. The later may include links to its web interface, some indications on controller reachability, or additional services provided by the server.
4. If the controller is part of a community network which runs some web application or database to organize their equipment, you may enter the URIs of the server in them under section *Community host*.
5. Click on *Save* at the bottom of the page.

API endpoints require more attention. At least a Registry API endpoint reachable from the management network should be declared. The installation creates two example endpoints for the Registry API and Controller API. To complete them:

1. Go to this controller's server page in the Controller web interface.
2. Under section *Server APIs*, change the *Base URI* for both endpoints from the example `https://[2001:db8:cafe::2]/api/` to `https://[MGMT_ADDR]/api/`, where `MGMT_ADDR` is the management address that appears under section *Management network* (e.g. `fdc0:7246:b03f::2` in IPv6 short form).
3. To get the certificate, log into the controller server (e.g. via SSH) and copy the content of `~confine/mytestbed/pki/cert` (use your own system user and testbed name if different). Paste it as the *Certificate* of both API endpoints.
4. Leave the API *Type* as is for both entries.
5. Leave the empty *Island* as is (it means that the endpoints are reachable through the management network).
6. Click on *Save* at the bottom of the page.

If the controller is reachable through other networks like the Internet or some community network, you may add more API endpoints using the *Add another Server API* link under section *Server APIs*. For each of those networks, select a different island (see Islands [A.4.1]) and specify the URI reachable from it (e.g. `https://controller.example.com/api/`), along with the proper type and certificate.

Similarly, you may also want to add new ***tinc* addresses** if the controller is connected to several networks:

1. Go to this controller's server page in the Controller web interface.
2. Click on *Manage tinc addresses* next to section *Tinc configuration* to get the list of tinc addresses for this server. An address without an explicit island is already present.
3. Click on the *Add tinc address* button to add a new address.
4. Provide an IP address (or DNS host name) and port (TCP and UDP) where *tinc* connections can be received.
5. Select the island the address belongs to (see Islands [A.4.1]).

6. Choose this controller as the host providing this address.
7. Click on *Save*.

Please note that this does not change the actual configuration of the tinc *daemon, which is setup by default to listen on port 655 of all interfaces.*

A.4.3. Sliver templates

You need to define at least one template that slice administrators can choose for their slices or particular slivers. Usually your testbed will only have **a few basic and well-maintained sliver templates** that can be reused for different applications and customized by slice administrators using sliver data. The more sliver templates your testbed supports, the more maintenance overhead they will require from testbed operators, and the more space they will take in nodes, possibly reducing sliver concurrence. The format and content of such templates depend on their *type* (which also implies how sliver data files are handled), and each template must indicate the node architectures it is compatible with.

The *Confined* release of Node software supports sliver templates running on *32-bit Linux kernels*, with template types `debian` or `openwrt` for Debian and OpenWrt slivers, respectively. In both cases, the template must be a plain gzip-compressed Tar archive (`*.tar.gz` or `*.tgz`) of the contents of a root file system. The template must be available to nodes over HTTP or HTTPS.

The archive format allows you to reuse existing images which you may easily find on the Internet:

- `rootfs` archives from OpenWrt downloads[8]
- OpenVZ precreated templates[9]
- `rootfs` archives from LXC pre-built containers[10] (after recompressing as gzip)

You may also build your own images:

- Using a real machine and backing up its root file system (not recommended).
- With `debootstrap` or similar to setup a Debian installation and customize it using `chroot`. When done, run `bsdtar -czf TEMPLATE-NAME.tar.gz --numeric-owner DIRECTORY`.
- Using LXC to create a container and customize it as a normal machine. When done, stop the container and run `bsdtar -czf TEMPLATE-NAME.tar.gz --numeric-owner -C /var/lib/lxc/NAME/rootfs .`
- Using VCT[11] and `vct_build_sliver_template TYPE VARIANT` (e.g. `debian wheezy`) to create a `vct-sliver-template-build-TYPE-VARIANT.tgz` archive (see Using the Virtual CONFINE Testbed[11] for usage instructions).

Once you have the sliver template in your computer, to make it available to nodes and slice administrators:

[8] https://downloads.openwrt.org/
[9] https://openvz.org/Download/template/precreated
[10] https://images.linuxcontainers.org/
[11] https://wiki.confine-project.eu/usage:vct

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Templates* icon to get the list of sliver templates in the testbed.
3. Click on the *Add template* button to register a new template.

Now you must provide the **configuration for the template**:

- A descriptive *name* that reveals relevant information to the user at first sight, e.g. *Debian 7 Wheezy i386 with Erlang*.
- Use the *description* field for a more detailed explanation of the template, like customizations on the base system. You may include links for further information.
- Select the *type* of the template as explained above, e.g. *Debian*.
- Choose all *node architectures* the template is compatible with, e.g. `i686` and `x86_64` for a template with Pentium Pro-compatible binaries on a testbed with 32-bit and 64-bit nodes. Since the *Confined* Node software only supports 32-bit kernels, you may only specify `i586` or `i686`.
- The *image file* is the archive that contains the actual sliver template, e.g. `/path/to/debian-wheezy-i386-erlang.tar.gz`. Image files uploaded via the Controller are served to nodes over the management network so that online nodes can always fetch them.
- The *image SHA256 checksum* helps nodes test that the image was correctly downloaded and it has not been corrupted or tampered. It is automatically computed by the Controller.
- Whether the template *is active*. Inactive templates can be kept for reference but they can not be used to deploy new slivers.

Click on *Save* when done. The image is uploaded and registered along the template.

A.4.4. Node image generation

Controller software includes a feature which allows the generation of system images for nodes, from a selectable base image and the registry and firmware configuration stored for the node by the Controller. The resulting images are **fully customized for their respective nodes**, so that little or no manual configuration should be necessary after installing them.

A base image is a single binary file which can be directly installed in the storage of the node device (see Installing a node via USB[12]) and contains its operating system. To make this possible, the (x86-compatible) image contains the Master Boot Record (MBR), the partition table and the required disk partitions and file systems (currently boot and root). The CONFINE Node software, based on OpenWrt[13], is installed in these partitions.

The customization of a base image for a node consists in making a copy of the base image, finding the partitions and mounting them locally, and then modifying the required files in place according to the node's configuration.

[12] https://wiki.confine-project.eu/usage:node-admin#installing_a_node_via_USB

A.4.4.1. Base images

For the firmware generation feature to work, you need to **upload at least one base image** to the Controller. Generally speaking, you need to configure one base image per node type you expect to have in the testbed, e.g. based on its architecture or purpose.

The CONFINE Project builds and distributes some images from the confine-dist repository[13] which are available in the CONFINE Node software repository[14]. Base images for the *Confined* release of Node software exist for two architectures: `i586` for single-core Pentium-compatible CPUs with at most 4 GiB RAM, and `i686` for single or multi-core Pentium Pro-compatible CPUs (SMP) with big memory support using PAE. The file `CONFINE-owrt-master-ARCH-current.img.gz` always points to the latest stable release for the `ARCH` architecture.

Once you have downloaded the base image to your computer, to add it to the Controller:

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Firmware config* icon to get to the firmware generator configuration page.
3. Click on the *Add another Base image* link under section *Base images*.
4. Give it a descriptive name, e.g. *Stable 2015-07-15 i686*.
5. Choose all node architectures the image is compatible with, e.g. `i686` for Pentium Pro and AMD64-compatible nodes. Since the *Confined* Node software only supports 32-bit kernels, you may only specify `i586` or `i686`.
6. Select the image file from your computer.
7. Choose whether this base image will be chosen by default when generating images if more than one image is suitable for a given node. The node administrator will always be able to choose, though.
8. Click on *Save* at the top or bottom of the form.

Once the image is uploaded and registered, it becomes available for firmware generation.

A.4.4.2. Firmware configuration

Besides the information about a node stored in the testbed registry, the Controller also keeps some specific firmware configuration for it (e.g. about networking). During firmware generation, the Controller uses this node-specific configuration along with some **generic settings** in order to customize a copy of the selected base image. These generic settings are also available in the firmware generator configuration page:

- *Image name* is a Python string template which allows you to customize the name of the resulting image files. For instance, `mytestbed-node-%(node_id)04d-%(arch)s.img.gz` may yield `mytestbed-node-1234-i686.img.gz`.
- *Config UCI* holds OpenWrt UCI[15] configuration options to be stored in the image. Their value is a Python expression where `node` is the Django model object for the node (see [14]).

[13] https://redmine.confine-project.eu/projects/confine/
[14] https://media.confine-project.eu/node/
[15] http://wiki.openwrt.org/doc/uci

- *Config files* allows you to generate arbitrary files in the image. Their path and content are Python expressions where `node` is the Django model object for the node, so you may generate several files with one expression. You can set their mode (as in `chmod MODE FILE`), and you can make them optional so that, during firmware generation, the node administrator can decide whether to include them or not (with a help text that you can customize under section *Config file help texts*).

As an example of a UCI option, you may configure an **alternative upgrade URL** for nodes to retrieve the base image when using the `confine.remote-upgrade` tool to upgrade the node system:

1. Go to the firmware generator configuration page in the Controller web interface.
2. Under section *Config UCI*, click on *Add another Config UCI* to add a new empy item.
3. Enter the section `node node` (i.e. file `/etc/config/node`, section `node`).
4. Enter the option name `latest_image_uri`.
5. Enter the new URL as the option value, e.g. `'https://[fdc0:7246:b03f::2]/base-images/current-i686.img.gz'` (use your own management address and mind the quotes) would make nodes request the image from this controller over the management network (of course, the controller's web server should be configured to serve those files).
6. Click on *Save* at the top or bottom of the form.

The option will be included in the images generated from this point on.

As an example of a configuration file, you may configure nodes to use the **controller NTP server** (which is automatically enabled during Controller installation) to synchronize the clocks of nodes with the controller's one:

1. Go to the firmware generator configuration page in the Controller web interface.
2. Under section *Config files*, click on *Add another Config File* to add a new empty item.
3. Enter the path `/etc/ntp.conf`.
4. Enter the content `'restrict default ignore\nrestrict 127.0.0.1\ndriftfile /var/lib/ntp/ntp.drift\nserver fdc0:7246:b03f:0:0:0:0:2 prefer iburst\n'` (use your own management address and mind the quotes).
5. Leave the default mode (which equals `0644`) and make the file not optional.
6. Click on *Save* at the top or bottom of the form.

The file will be included in the images generated from this point on.

A.4.4.3. Plugins

The Controller software allows the use of different plugins that affect the firmware generation process. To enable or disable them, go to the firmware generator configuration page in the Controller web interface and look under section *Plugins*:

- `PasswordPlugin` adds a `root` password input to the node firmware generation page. Otherwise, generated images only allow `root` access using SSH keys.

- `AuthKeysPlugin` adds an input to the node firmware generation page where additional SSH keys for `root` access to the node can be specified, e.g. for centralized maintenance by testbed operators (see Remote node maintenance [A.6.2]), other automated remote access, or for users not present as node administrators in the testbed registry.
- `USBImagePlugin` adds an option to the node firmware generation page to adapt the created image to be written to a USB drive and booted to perform the installation. This is especially useful for the initial preparation of nodes without an operating system or some easy way to install system images.

Check or uncheck the box next to each plugin and click on *Save* when done.

If you enable `USBImagePlugin` you will need to download and place the CONFINE installation image[16] in the Controller deployment directory. For instance, via SSH (use your own system user and testbed name if different):

```
$ ssh confine@controller.example.com  # in your computer
$ wget -P ~/mytestbed \
  "https://media.confine-project.eu/confine-install/confine-install.img.gz" \
  # in the controller
```

A.4.4.4. Installing new plugins

To be able install your own firmware generation plugins, you need to add a new application in the Controller to export them. For instance, let us call the application `firmwareplugins`. Use the system user to create it in the Controller deployment directory as a Python package directory (use your own testbed name if different):

```
$ mkdir ~/mytestbed/firmwareplugins
$ touch ~/mytestbed/firmwareplugins/__init__.py
```

Now add the application by appending the following lines to `~/mytestbed/mytestbed/settings.py`:

```
# Enable custom firmware plugins
from controller.utils.apps import add_app
INSTALLED_APPS = add_app(INSTALLED_APPS, 'firmwareplugins')
```

To install a new firmware generation plugin (or upgrade an existing one), drop its Python code under the `firmwareplugins` directory and import its name:

```
$ cp /path/to/myplugin.py ~/mytestbed/firmwareplugins
$ echo 'from .myplugin import MyPlugin' \
  >> ~/mytestbed/firmwareplugins/__init__.py
```

You also need to synchronize the plugins with the database and restart Controller services for the changes to take effect:

```
$ python ~/mytestbed/manage.py syncfirmwareplugins
$ sudo python ~/mytestbed/manage.py restartservices
```

An entry for enabling the new plugin will appear under section *Plugins* in the firmware generator configuration page in the Controller web interface.

[16] https://media.confine-project.eu/confine-install/confine-install.img.gz

A.4.5. Services in the management network

As noted in the example node configuration file described in Firmware configuration [A.4.4.2], you may run arbitrary services on the management network that will be available to all testbed servers, nodes, slivers and hosts. It is not necessary that these services run on servers operated by testbed administrators, in fact they may run on any server, node, sliver or host connected to the management network.

In case you want to dedicate some host to a specific, long-running service, it is recommended that you register it as a **testbed host** (see Adding a host to the management network[17]) and introduce the service in its description field. This can actually be done by any testbed user without special privileges.

You may use a service like this for core testbed support for whoever wants to trust it, e.g. to provide some proxy cache service to the Registry API for a set of nodes with bad connectivity (just remember to provide the proper URI and certificate when generating node firmware).

A.4.6. Adding a management network gateway

Some nodes in your testbed may be unable to reach the testbed registry or the controller by themselves (e.g. some community networks lack native access to the Internet). In this case you may pick a server which does have access both to the controller and those nodes, and configure it in the testbed as a **management network gateway server** (or *gateway server* for short), so that it extends the testbed management network to the nodes and makes them and their slivers available in the testbed (see [8]).

In the setup described below, the gateway server will connect its *tinc* daemon to the controller, and nodes will connect theirs to the gateway instead of the controller. At the gateway, the scripts from CONFINE utilities[15] will be used to periodically refresh the information of hosts allowed to connect to its *tinc* daemon.

A.4.6.1. Gateway server registration

First you need to add the new server to the testbed registry:

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Servers* icon to get the list of servers in the testbed.
3. Click on the *Add server* button to register a new server.
4. Provide an informative name (e.g. *My Network gateway*) and a description.
5. Click on *Save and continue editing*.
6. Note down the IPv6 address under section *Management network*, i.e. its **management address** (e.g. `fdc0:7246:b03f:0:0:0:0:3`).
7. Click on *Manage tinc addresses* next to section *Tinc configuration* and add at least one address which is reachable from the nodes. When choosing this server as the host of the *tinc* address, please note down the name outside of the parentheses, i.e. its ***tinc* name** (e.g. `server_3`).

[17] https://wiki.confine-project.eu/usage:common#adding_a_host_to_the_management_network

Now you need to get some configuration information about the controller server you want the gateway's *tinc* daemon to connect to (you may repeat these steps if you want to connect to other servers as well):

1. Go to that server's page in the Controller web interface.
2. Note down the IPv6 address under section *Management network*, i.e. its **management address** (e.g. `fdc0:7246:b03f:0:0:0:0:2`).
3. Note down the public key under section *Tinc configuration*, i.e. its **tinc public key**.
4. Click on *API* on the menu bar. You are presented with the Registry API representation of the server.
5. Note down the string next to the `tinc/name` JSON member, i.e. its ***tinc* name** (e.g. `server_2`).
6. Note down some addresses under the `tinc/addresses` JSON member, i.e. its ***tinc* addresses**, which are reachable from the gateway server (e.g. `controller.example.com 655`).

You will also need the **Registry API base URI** to retrieve *tinc* host information from, which usually is one of the URIs published by the same server (visible under section *Server APIs* of its server page in the Controller web interface, e.g. `https://controller.example.com/api/`).

A.4.6.2. Gateway tinc configuration

Now you can proceed to setup your server. Here we assume a Debian installation, but any *tinc*-compatible Unix box (physical or virtual) with Git, Python and a Cron daemon should suffice for this setup. First we will configure the *tinc* daemon the nodes will connect to.

1. Log into the gateway server (e.g. via SSH) as `root`.
2. Install *tinc*: `apt-get install tinc`.
3. Create directories for your testbed *tinc* configuration and host files, e.g. `mkdir -p /etc/tinc/mytestbed/hosts`.
4. Create the *tinc* configuration file with your gateway server's *tinc* name to connect to the controller's *tinc* name, for instance:

```
# cat /etc/tinc/mytestbed/tinc.conf
Name = server_3
ConnectTo = server_2
StrictSubnets = yes
```

5. Create the controller's *tinc* host file with its management address, *tinc* addresses and public key:

```
# cat /etc/tinc/mytestbed/hosts/server_2
Subnet = fdc0:7246:b03f:0:0:0:0:2/128
Address = controller.example.com 655
-----BEGIN PUBLIC KEY-----
[...]
-----END PUBLIC KEY-----
```

6. Create this server's *tinc* host file with its management address and generate its key pair:

```
# echo 'Subnet = fdc0:7246:b03f:0:0:0:0:3/128' \
  > /etc/tinc/mytestbed/hosts/server_3
# tincd -n mytestbed -K  # accept the default files
```

7. Create a script for setting the management network interface and its address up, and make it executable:

```
# cat /etc/tinc/mytestbed/tinc-up
#!/bin/sh
ip -6 link set "$INTERFACE" up mtu 1400
ip -6 addr add fdc0:7246:b03f:0:0:0:0:3/48 dev "$INTERFACE"
# chmod a+rx /etc/tinc/mytestbed/tinc-up
```

8. Configure the network to be enabled on boot and restart *tinc*:

```
# echo mytestbed >> /etc/tinc/nets.boot
# service tinc restart
```

9. Finally, copy this server's public key block from the *tinc* host file you created before (`/etc/tinc/mytestbed/hosts/server_3`), paste it in the *Public Key* box under section *Tinc configuration* in this server's page in the Controller web interface and click on *Save*.

From this point on, the gateway server is allowed in the testbed management network (but see issue #694[18]). You should be able to successfully ping the controller's management address (e.g. `ping6 fdc0:7246:b03f::2`).

A.4.6.3. Periodic refresh of tinc hosts

Now we shall configure a Cron task to periodically update *tinc* host files with the information retrieved from the testbed registry, using the gateway scripts available at the `confine-utils` repository.

1. Clone the `confine-utils` repository to `/opt`:

```
# git clone \
  http://git.confine-project.eu/confine/confine-utils.git \
  /opt/confine-utils
```

2. Use Python Setuptools to install the latest Requests library:

```
# apt-get install python-setuptools
# easy_install --upgrade requests
```

3. Test if you are able to retrieve *tinc* hosts information from the testbed registry using the `fetch_tinc_hosts.py` script, for instance:

```
# UTILS_PATH=/opt/confine-utils
# REGISTRY_URI=https://controller.example.com/api/
# cd $(mktemp -d)
# env PYTHONPATH=$UTILS_PATH \
  python $UTILS_PATH/gateway/fetch_tinc_hosts.py "$REGISTRY_URI"
```

4. Create the `/etc/cron.d/local-mytestbed` crontab to run the `refresh-tinc-hosts` script against the registry once per hour (use your own testbed name and registry URI):

[18] https://redmine.confine-project.eu/issues/694

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
TINC_DIR=/etc/tinc/mytestbed
REGISTRY_URI=https://controller.example.com/api/
42 * * * *  root  /opt/confine-utils/gateway/refresh-tinc-hosts "$TINC_DIR" "$REGISTRY_URI"
```

5. Reload Cron with `service cron reload`.

As soon as the task runs for the first time and information is collected, *tinc* connections from testbed servers, nodes and hosts will be allowed.

Node admins who want to use this server as a management network gateway may select it in the Default connect to *drop-down box under section* Tinc configuration *of the node's page in the Controller web interface, and then regenerate node firmware. Otherwise they may tune the* `registry.base_uri` *and* `registry.cert` *UCI options of file* `/etc/config/confine` *in OpenWrt.*

A.5. Managing users and groups

A.5.1. User registration

The Controller has three modes of handling the registration of new user accounts:

- With **open registration**, everybody can create an account and use it without previous permission or notification, only email address validation is required.
- With **restricted registration**, everybody can create an account but some operator should approve it before it can be used to log in.
- With **closed registration**, only existing operators can create new accounts.

These modes are selectable via the `USERS_REGISTRATION_MODE` setting (see Controller configuration [A.3]). See Registering in a controller[19] for the user's perspective on account registration.

When **restricted registration** is active and someone registers an account, after validating its email address a notification is sent to the approval email address configured in the `EMAIL_REGISTRATION_APPROVE` setting. To approve (i.e. enable) the account:

1. Follow the link contained in the email to the user page in the Controller web interface.
2. Check the *Is active* box.
3. Click on *Save*.

A message is sent to the approval address notifying the activation of the account. The new user is notified via email as well.

When **closed registration** is active, user accounts must be created manually:

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Users* icon to get the list of users in the testbed.
3. Click on the *Add user* button to register a new user.
4. Provide an id-like user name, a password, a full name and an email address.

[19] https://wiki.confine-project.eu/usage:common#registering_in_a_controller

5. Click on *Save*.

The user is created and activated, and no notifications are sent to operators or the user.

For more details on user registration, see [16].

A.5.2. Managing superusers

Superusers are normal users which have been granted special administrative powers by other existing superusers. Superusers have unrestricted access to the testbed registry (e.g. they can grant node and slice management permissions to groups, see [17] for more details), and they can also administer the Controller itself (e.g. by uploading node base images or managing tasks).

To convert a normal user in a superuser:

1. Log into the Controller web interface. You are presented with the *dashboard*.
2. Click on the *Users* icon to get the list of users in the testbed. Look for the affected user and enter its page.
3. Check the *Is superuser* box.
4. Click on *Save*.

Removing superuser powers can be done in the same way by the user itself of by another superuser.

> *Please note that superusers do not get Controller error or account approval notifications by default. For that, use the* `ADMINS` *and* `EMAIL_REGISTRATION_APPROVE` *settings, respectively (see Controller configuration [A.3]).*

A.5.3. Group permissions

Although any registered user can create a group and automatically become its group administrator (see Creating a group[20]), only groups with the proper permissions can create nodes or slices associated with them. In the *Confined* release of Controller software, these permissions are disabled by default for new groups and can only be granted by superusers.

When a group administrator requests node or slice creation permission for the group (see Applying for slice or node management[21] for the user's perspective), an email message is sent to superusers notifying about the request for node or slice resources. To allow or decline the request:

1. Follow the link contained in the email to the group page in the Controller web interface.
2. Check or uncheck the *Allow nodes* or *Allow slices* box.
3. Click on *Save*.

[20] https://wiki.confine-project.eu/usage:group-admin#creating_a_group
[21] https://wiki.confine-project.eu/usage:group-admin#applying_for_slice_or_node_management

This resolves the requests. Granting group permissions can be undone in the same way only by superusers.

See Resource Management (only superusers)[22] for more details on group permissions.

A.5.4. Customizing registration templates

The user registration procedure includes sending several emails (e.g. how to validate the user's email address, next steps, etc.). Although the Controller software provides a complete set of templates, you can customize them following these instructions:

1. Log into the controller server (e.g. via SSH) as the system user and create a **custom templates directory** under your Controller deployment directory (e.g. `~confine/mytestbed/templates`, use your own system user and testbed name if different).
2. Write your customized templates. You can use the Controller default registration templates[23] as examples.
3. Put the customized templates in your custom templates directory. Please keep the directory structure and filenames of the default ones:

```
$ ls -R1 ~/mytestbed/templates  # example with all templates customized
/home/confine/mytestbed/templates/templates:
registration
/home/confine/mytestbed/templates/templates/registration:
activation_email_subject.txt  # ask user for email validation (email subject)
activation_email.txt          # ask user for email validation (email body)
activation_complete.html      # user validated the email address, moderation pending
registration_complete.html    # user completed the registration procedure
registration_closed.html      # information page when registration is closed
```

4. Extend your `TEMPLATE_DIRS` setting with the custom templates directory and restart testbed services (see Controller configuration [A.3]):

```
TEMPLATE_DIRS = ('/home/confine/mytestbed/templates', ) + TEMPLATE_DIRS
```

A.6. Testbed maintenance

As a testbed operator, there are some tasks that you are likely to come across to keep the testbed working properly. Here you will find some recommendations to maintain the controller, apply some changes massively to the nodes, or tools to debug possible trouble.

A.6.1. Upgrading Controller to a newer version

From time to time a new version of the Controller software is released. It is announced on the `confine-devel` mailing list[10] and the release changes are published on [18].

To **check the current version** of your Controller log into the controller server (e.g. via SSH) as the system user and run the following command (use your own system user and testbed name if different):

[22] https://wiki.confine-project.eu/soft:server-user-manual#resource_management_only_superusers

[23] https://redmine.confine-project.eu/projects/controller/repository/revisions/master/show/controller/templates/registration

```
$ python ~/mytestbed/manage.py controllerversion
1.0.1
```

Then, you can **check if you have the latest version** by running:

```
$ pip search confine-controller
confine-controller     - Django-based framework for building control servers for
                         computer networking and distributed systems testbeds.
  INSTALLED: 1.0.1
  LATEST:    1.0.2
```

If you want **to upgrade**, check [18] for the latest version and then run:

```
$ sudo python ~/mytestbed/manage.py upgradecontroller
```

In a few minutes you will have your Controller up to date.

A.6.2. Remote node maintenance

Controller provides via the Maintenance application[19] the mechanism to perform maintenance operations remotely in a set of nodes of the testbed.

If you go to *Administration* > *Maintenance* > *Operation* (e.g. `http://controller.example.com/admin/maintenance/operation/`), you can create a new **operation** (e.g. a script to fix a wrong configuration), select the nodes where you want to run it and check the status of the execution.

Please note that node administrators can choose to allow or not the centralized access to their nodes, which means that the Controller will only be able to perform maintenance operations in the nodes which accept its SSH key.

A.6.3. Testbed status monitoring

The Controller provides several monitoring tools which can help testbed operators to detect and debug problems and also provide an overview of the health of the testbed. There are **three Controller applications in charge of collecting information** about the current status of the testbed, one of them monitors the Controller itself (see [12]) and the others keep track of the other components of the testbed (see [20, 21]).

Besides the tools provided by these applications, the Controller provides a set of reports which provide a general overview of the testbed:

- Slices allocation per group in the testbed: *Slices* > *Slices* > *Status overview* (e.g. `http://controller.example.com/admin/state/state/slices/`)
- Slivers allocation by node in the testbed: *Slices* > *Slivers* > *Status overview* (e.g. `http://controller.example.com/admin/state/state/slivers/`)
- Testbed status report: *Nodes* > *Summary* (e.g. `http://controller.example.com/admin/state/state/report/`)
- Map of the testbed: *Nodes* > *Nodes Map* (e.g. `http://controller.example.com/gis/map/`)

A.6.3.1. Monitor application

The Monitor application[12] collects information about the status of the server where the Controller is running: memory usage, CPU load, storage, bandwidth, etc. You can get an overview of the historical resources usage by clicking on the `State` link of the main server: *Nodes* > *Server* > *Main server* (e.g. `http://controller.example.com/admin/nodes/server/monitor/`).

A.6.3.2. State application

The State application[20] retrieves information about nodes and slivers via the Node API[7]. The information retrieved is shown as a summary in the node list (e.g. `http://controller.example.com/admin/nodes/node/`) and sliver list (e.g. `http://controller.example.com/admin/slices/sliver/`), displaying states such as `OFFLINE`, `PRODUCTION`, or `STARTED`. You can get detailed information by clicking on them or going to the state page of a node or sliver (e.g. `http://controller.example.com/admin/nodes/node/1/state`). Other information provided is the node firmware version, so you can know what version of the CONFINE node system[22] the nodes are running.

A.6.3.3. Pings application

The Pings application[21] checks the connectivity through the management network[8]. Periodically, the controller performs a ping to the other components of the testbed (nodes, hosts and slivers). Go to the *State* page of the component and then click on the *Pings* button of the action links (in the top right corner) to see the result of the last pings.

A.6.4. Controller logs

You can find the logs related to the controller environment in `/var/log/` (use your own system user and testbed name if different):

- NginX (web server): `/var/log/nginx/[access|error].log`. These can grow unwieldy in big testbeds, so you may want to set `access_log off` in the `/api` locations of your configuration (e.g. `/etc/nginx/conf.d/mytestbed.conf`).
- *tinc* (management network overlay): `grep tinc.mytestbed /var/log/syslog`
- Celery (task queue): `/var/log/celery/`
  - `w1` & `w2`: Celery workers
  - `beat`: periodic task executer
  - `celeryev`: celery monitor

A.6.5. Controller Celery tasks

Celery[23] is a distributed task queue used by the Controller to execute tasks in an asynchronous way (e.g. firmware generation, monitoring tasks...). The admin site provides an

interface to manage tasks run by Celery workers. You can access it via *Administration* > *Djcelery* > *Tasks* (e.g. `http://controller.example.com/admin/djcelery/taskstate/`). There you get an overview of the tasks handled by Celery and their current state, which may help you to debug failures (the task state may include a traceback).

A.6.6. Controller disk usage

The Controller has a complex environment which involves several services (Celery, RabbitMQ, PostgreSQL...), all of them having its own particularities. As a testbed operator you should take care about all of them to avoid exhausting disk space.

One of the known issues when there is not enough free disk space (less than 1 GiB) is that the RabbitMQ[24] (messaging system used by the Controller to communicate with Celery) daemon stops (see RabbitMQ disk alarms[24] for more details).

The following subsections provide some recommendations to keep your disk clean.

A.6.6.1. Firmware generation temporary files

During firmware generation, the Controller creates a temporary directory `/tmp/tmpXXXXXX` to unpack and customize the base image. In some situations, the Controller may not clean up the workspace and leave this temporal directory behind.

```
$ ls /tmp/
tmprJlfI_
$ du -hs /tmp/tmprJlfI_
257M /tmp/tmprJlfI_
```

You can safely remove them if they are older than one day.

A.6.6.2. Clean orphan files

Base images for firmware, slices and sliver templates or firmware images need a lot of disk space and can raise *insufficient disk space* errors (see issue #326[25]).

The controller provides a periodic task that automatically removes old files, but an extra configuration step is required: **enable clean orphan files task**. To keep clean the filesystem when files associated to a model are not longer necessary (e.g. firmware builds after its node's deletion), the Controller provides a periodic task (disabled by default) that deletes those files (see related issue #192[26]).

This task requires the `django-orphaned` app installed (as `root`):

```
# pip install https://github.com/ledil/django-orphaned/archive/master.zip
```

Add `django-orphaned` to `INSTALLED_APPS` in the Controller settings file (see Controller configuration [A.3]) and configure the cleanup apps:

[24] https://www.rabbitmq.com/disk-alarms.html
[25] http://redmine.confine-project.eu/issues/326
[26] http://redmine.confine-project.eu/issues/192

```
INSTALLED_APPS = (
    'django_orphaned',
    ...
)
from os import path
from firmware.settings import FIRMWARE_BUILD_IMAGE_PATH, FIRMWARE_BASE_IMAGE_PATH
from slices.settings import (SLICES_TEMPLATE_IMAGE_DIR,
    SLICES_SLICE_DATA_DIR, SLICES_SLIVER_DATA_DIR)
FW_BUILD_IMAGE_ROOT = os.path.join(PRIVATE_MEDIA_ROOT , FIRMWARE_BUILD_IMAGE_PATH)
FW_BASE_IMAGE_ROOT = os.path.join(MEDIA_ROOT, FIRMWARE_BASE_IMAGE_PATH)
SLICES_TEMPLATE_ROOT = os.path.join(MEDIA_ROOT, SLICES_TEMPLATE_IMAGE_DIR)
SLICES_DATA_ROOT = os.path.join(MEDIA_ROOT, SLICES_SLICE_DATA_DIR)
SLIVER_DATA_ROOT = os.path.join(MEDIA_ROOT, SLICES_SLIVER_DATA_DIR)
ORPHANED_APPS_MEDIABASE_DIRS = {
    'firmware': {
        'root': (FW_BUILD_IMAGE_ROOT, FW_BASE_IMAGE_ROOT),
        'exclude': ('.gitignore')
    },
    'slices': {
        'root': (SLICES_TEMPLATE_ROOT, SLICES_DATA_ROOT, SLIVER_DATA_ROOT),
        'exclude': ('.gitignore')
    },
}
```

You can check if is properly configured by running:

```
$ python ~/mytestbed/manage.py deleteorphaned --info
```

A.6.7. Controller database

The monitoring applications of the Controller (see [21, 20]) make an intensive usage of the database (they periodically store monitoring data there). Although the Controller implements mechanisms to reduce the disk usage footprint by aggregating old data, in some situations the database may become huge and some write access operations very slow or even fail.

Here you can find some tips to monitor and optimize the database:

- Adjust the aggregation periods[27] used for downsampling the pings (according to the accuracy you want for older pings) in the Controller settings file (see Controller configuration [A.3]):

  ```
  PING_DEFAULT_INSTANCE['downsamples'] = (
      # Limitations: you can not say 16 months or 40 days
      #              but you can say 2 years or 2 months
      # pings older than 1 year aggregates as 4 hour samples
      (relativedelta(years=1), timedelta(minutes=240)),
      # pings older than 6 months aggregates as 1 hour samples
      (relativedelta(months=6), timedelta(minutes=60)),
      # pings older than 2 weeks aggregates as 5 minutes samples
      (relativedelta(weeks=2), timedelta(minutes=5)),
  )
  ```

- Check a database size and consider removing very old data to reduce its size:

  ```
  $ psql controller
  controller=> select pg_size_pretty(pg_database_size('controller'));
   pg_size_pretty
  ----------------
  ```

[27] https://wiki.confine-project.eu/soft:server-apps-pings#settings

```
 529 MB
(1 row)
controller=> SELECT nspname || '.' || relname AS "relation",
  pg_size_pretty(pg_relation_size(C.oid)) AS "size"
  FROM pg_class C
  LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
  WHERE nspname NOT IN ('pg_catalog', 'information_schema')
  ORDER BY pg_relation_size(C.oid) DESC
  LIMIT 5;
                    relation                    |  size
------------------------------------------------+---------
 public.monitor_timeserie                       | 216 MB
 public.monitor_timeserie_name_42a3bc16d24d56d6 | 101 MB
 public.monitor_timeserie_pkey                  | 36 MB
 public.pings_ping                              | 34 MB
 public.pings_ping_date_f29a98176e19536         | 32 MB
```

- Execute operations to optimize tables like VACUUM[28] and REINDEX[29].

A.6.8. Caching API requests

A testbed with many nodes, slivers and users may result in high controller CPU usage and long response times while replying to API requests. The web server can be configured to perform some caching if the controller has enough free memory, so that the response for some requests can be temporarily stored and served quickly with low delay and CPU usage.

An example caching configuration for NginX in the `/etc/nginx/conf.d/mytestbed.conf` file may include the following options for an in-memory (`/dev/shm/nginx`) 50 MiB storage:

```
# Define an in-memory cache storage named ``cache`` with 50 MiB.
proxy_cache_path /dev/shm/nginx levels=1:2 keys_zone=cache:50m;
server {
    listen [fdc0:7246:b03f::2]:443 ssl;  # cache mgmt net requests...
    [...]
    location /api/ {  # ...for the API
        [...]
        proxy_cache       cache;
        proxy_cache_key   $host$uri$is_args$args$http_accept_encoding$http_accept;
        proxy_cache_valid 1m;  # keep entries for at most 1 month
        expires           1m;
        set $skip_cache 0;
        if ($request_method != GET) {  # only cache GET requests...
            set $skip_cache 1;
        }
        if ($http_cookie) {  # ...without cookies...
           set $skip_cache 1;
        }
        if ($http_authorization) { # ...and anonymous (like most nodes')
           set $skip_cache 1;
        }
        proxy_cache_bypass  $skip_cache;
        [...]
    }
```

Restart NginX when done with `service nginx restart`.

[28] http://www.postgresql.org/docs/current/static/sql-vacuum.html
[29] http://www.postgresql.org/docs/current/static/sql-reindex.html

A.7. Troubleshooting

Here you may find ways to fix some common known issues with software used to run a CONFINE testbed.

A.7.1. Problem with generated certificates

As discussed on issue #625[30], there is a firmware generation bug that affects the generation of certificates used by `uhttpd` (node web server). Although it is fixed on Controller version 0.11.7[31], operators of existing testbeds need to perform a few actions in their Controller as the system user.

1. Upgrade the Controller to version `0.11.7` or later (see Upgrading Controller to a newer version [A.6.1]):

   ```
   $ sudo python ~/mytestbed/manage.py upgradecontroller \
     --controller_version=0.11.7
   ```

2. Patch the controller server API certificate in the testbed registry:

   ```
   $ # Get path of server certificate.
   $ python ~/mytestbed/manage.py print_settings | grep PKI_CA_CERT_PATH
   PKI_CA_CERT_PATH                        = '/var/lib/vct/server/pki/ca/cert'
   $ # Backup current certificate.
   $ mv ~/mytestbed/pki/ca/cert ~/mytestbed/pki/ca/cert.old
   $ # Show current certificate information (keep it to generate new certificate).
   $ openssl x509 -in ~/mytestbed/pki/ca/cert.old -text
   [...]
   $ # Generate a new certificate with version 3 (0x2).
   $ python ~/mytestbed/manage.py setuppki  # include your organization details
   ```

3. Remove the invalid node certificate from `NodeKeys` (NOT from `node@registry:/api/cert`) using `python ~/mytestbed/manage.py shell_plus`:

   ```
   from M2Crypto import RSA, X509
   def get_node_certificate_version(node):
       """Check if certificate has invalid version (0x3)."""
       if node.keys.cert is None:
           return False
       pem_string = str(node.keys.cert)
       cert = X509.load_cert_string(pem_string)
       return cert.get_version()
   def is_valid_node_certificate_version(node):
       if  get_node_certificate_version(node) == 3:
           return False
       return True
   def fix_node_certificate_version(node):
       """
       Remove invalid stored /etc/uhttpd.crt.pem file
       Will be regenerated on next firmware build (node.api.cert too)
       NOTE: should be executed with patched controller (X509 version 0x2)
       """
       assert not is_valid_node_certificate_version(node)
       cert = node.files.get(path=NodeKeys.CERT)
       assert cert.content == node.api.cert, "Node %s" % node.pk
       cert.delete()
   # Get nodes with invalid certificate version.
   affected_nodes = []
   for node in Node.objects.all():
       if not is_valid_node_certificate_version(node):
           affected_nodes.append(node.pk)
           #fix_node_certificate_version(node) # UNCOMMENT to massive fix
   print "Fixed %i" % len(affected_nodes)
   ```

[30] https://redmine.confine-project.eu/issues/625
[31] https://wiki.confine-project.eu/soft:server-release-notes#section0117

4. Upgrade the affected nodes (see Node upgrade[32]).

A.7.2. Celery tasks not shown on Django admin interface

If you go to *Administration* > *Djcelery* > *Tasks* and the lists show no objects or only old tasks (e.g. received yesterday, 2 days ago...), you need to check if the Celery monitor components are running:

- `celeryev` (e.g. `ps ax | grep celeryev`)
- `celerybeat` (e.g. `ps ax | grep celerybeat`)

If any of them is not running, start it as `root` with `service SERVICE start` (with `SERVICE` being `celeryevcam` or `celerybeat`). Otherwise you may try restarting it with `service SERVICE restart`.

A.7.3. Uploading templates or other big files fails

Probably you have reached NginX's `POST` maximum size. This limit exists for discouraging the upload of big sliver templates (because they are supposed to be transferred over not that reliable community networks).

You should remove or increase `client_max_body_size` in your NginX configuration (it may appear more than once).

A.7.4. Celery task fails with "Too many open files"

The operating system has a limit of open files for processes. As the ping and state apps use one file descriptor per node and sliver for a big number of nodes and slivers, the limit of open files can be reached (e.g. 240 nodes and 800 slivers means 1040 open files). In this situation Celery tasks show the `FAILED` state and this error message:

```
OperationalError: could not create socket: Too many open files
```

You can check the limit by running:

```
$ ulimit -Hn  # hard limit
4096
$ ulimit -Sn  # soft limit
1024
```

To temporally increase this limit you can run as `root`:

```
# ulimit -Sn 2048
# ulimit -Sn
2048
```

To make this change permanent in Celery, place the previous commands in Celery's initialization script and restart the daemon. See the Debian Wiki page on limits[33] for more information.

[32] https://wiki.confine-project.eu/soft:node-upgrade
[33] https://wiki.debian.org/Limits

A.7.5. Other issues

If you have found an issue that is not solved here, and you have not seen any other related information to it in the CONFINE wiki[25] or CONFINE's Redmine site[26], consider asking for help in the `confine-devel` mailing list[10].

Acronyms

Y3 the third year of the CONFINE project (Oct 2013–Sep 2014)

Y4 the fourth year of the CONFINE project (Oct 2014–Sep 2015)

Bibliography

[1] "Community-Lab portal." [Online]. Available: https://community-lab.net/ 2, A.1

[2] "Community-Lab/CONFINE administrator's guide." [Online]. Available: https://wiki.confine-project.eu/admin:start 2

[3] "Community-Lab/CONFINE user's guide." [Online]. Available: https://wiki.confine-project.eu/usage:start 2, A.1

[4] "CONFINE Project portal." [Online]. Available: https://confine-project.eu/ A.1

[5] "CONFINE Controller software." [Online]. Available: https://wiki.confine-project.eu/soft:server A.1, A.2.2

[6] "CONFINE testbed architecture." [Online]. Available: https://wiki.confine-project.eu/arch:start A.1

[7] "CONFINE REST API." [Online]. Available: https://wiki.confine-project.eu/arch:rest-api A.1, A.6.3.2

[8] "CONFINE management network architecture." [Online]. Available: https://wiki.confine-project.eu/arch:management-network A.1, A.2.1, A.4.6, A.6.3.3

[9] "Addressing in CONFINE." [Online]. Available: https://wiki.confine-project.eu/arch:addressing A.2.1

[10] "CONFINE development mailing list." [Online]. Available: http://lists.confine-project.eu/mailman/listinfo/confine-devel A.2.2, A.6.1, A.7.5

[11] "Virtual CONFINE Testbed (VCT)." [Online]. Available: https://wiki.confine-project.eu/soft:vct A.2.2, A.4.3

[12] "CONFINE Controller Monitor application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-monitor A.3, A.6.3, A.6.3.1

[13] "OpenWrt, a Linux distribution for embedded devices." [Online]. Available: https://openwrt.org/ A.4.4

[14] "CONFINE Controller Nodes application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-nodes A.4.4.2

[15] "CONFINE utilities." [Online]. Available: https://wiki.confine-project.eu/soft:utilities A.4.6

[16] "CONFINE Controller Registration application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-registration A.5.1

[17] "Roles and permissions in CONFINE." [Online]. Available: https://wiki.confine-project.eu/arch:roles-permissions A.5.2

[18] "CONFINE Controller release notes." [Online]. Available: https://wiki.confine-project.eu/soft:server-release-notes A.6.1

[19] "CONFINE Controller Maintenance application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-maintenance A.6.2

[20] "CONFINE Controller State application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-state A.6.3, A.6.3.2, A.6.7

[21] "CONFINE Controller Pings application." [Online]. Available: https://wiki.confine-project.eu/soft:server-apps-pings A.6.3, A.6.3.3, A.6.7

[22] "CONFINE Node software." [Online]. Available: https://wiki.confine-project.eu/soft:node A.6.3.2

[23] "Celery distributed task queue." [Online]. Available: http://www.celeryproject.org/ A.6.5

[24] "RabbitMQ messaging system." [Online]. Available: https://www.rabbitmq.com/ A.6.6

[25] "CONFINE Project wiki." [Online]. Available: https://wiki.confine-project.eu/ A.7.5

[26] "CONFINE Project Redmine site." [Online]. Available: https://redmine.confine-project.eu/ A.7.5

The CONFINE project

September 2015

CONFINE-201510-D3.4-1.2