The sip:provider CE Handbook

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

1.1 About this Document

This document describes the architecture and the operational steps to install, operate and modify the Sipwise sip:provider Community Edition (CE).

The first chapter gives a general overview of the CE. It describes what it is, what it contains and who should use it.

The second chapter guides through the installation process of the CE. It lines out the prerequisites and the steps required to install it.

The third chapter describes the steps to configure the CE in order to offer VoIP services to end users.

The fourth chapter shows the different customer self-care interfaces and describes how to configure them.

The fifth chapter describes the billing interface, so you can rate calls and export call detail records.

The sixth chapter describes in detail the steps necessary if you want to start modifying local configuration files to adapt and extend the system.

The seventh chapter guides through the provisioning interface (SOAP and XMLRCP) system.

The eighth chapter describes best practices for production systems regarding system security and backup/restore procedures.

More chapters will be added in the future to document every aspect of the system.

1.2 Getting Help

1.2.1 Community Support

We have set up the *spce-user* mailing list, where questions are answered on a best-effort basis and discussions can be started with other community users.

1.2.2 Commercial Support

If you need professional help setting up and maintaining the CE, send an email to support@sipwise.com.

Sipwise also provides training and commercial support for the platform. Additionally, we offer a migration path to the sip:provider PRO appliance, which is the commercial, carrier-grade version of the CE. If the user base grows on the CE, this will allow operators to migrate seamlessy to a highly available and scalable platform with defined service level agreements, phone support and on-call duty. Please visit www.sipwise.com for more information on commercial offerings.

1.3 What is sip:provider CE?

The CE is a SIP based Open Source Class5 VoIP soft-switch platform providing rich telephony services. It offers a wide range of features to end users (call forwards, voicemail, conferencing, call blocking, click-to-dial, call-lists showing near-realtime accounting information etc.), which can be configured by them using the customer-self-care web interface. For operators, it offers a fully web-based administrative panel, allowing them to configure users, peerings, billing profiles etc., as well as viewing real-time statistics of the system. For thight integration into existing infrastructures, it provides SOAP and XMLRPC APIs.

The CE can be installed in a few steps within a couple of minutes and requires no knowledge about configuration files of specific software components.

1.4 What is inside the sip:provider CE?

Unless other free VoIP software, the CE is not a single application, but a whole software platform, the Sipwise NGCP (Sipwise Next Generation Communication Platform), which is based on Debian GNU/Linux.

Using a highly modular design approach, the NGCP leverages popular open-source software like MySQL, Apache, Catalyst, Kamailio, SEMS, Asterisk etc. as its core building blocks. These blocks are glued together using optimized and proven configurations and work-flows and are complemented by building blocks developed by Sipwise to provide fully-featured and easy to operate VoIP services.

After downloading and starting the installer, it will fetch and install all the required Debian packages from the relevant Debian repositories. The installed applications are then managed by the NGCP Configuration Framework, which allows to change system parameters in a single place, so administrators don't need to have any knowledge of the dozens of different configuration files of the different packages. This provides a very easy and bullet-proof way of operating, changing and tweaking the otherwise quite complex system.

Once installed and configured, integrated web interfaces are provided for both end users and administrators to use the CE. By using the provided provisioning and billing APIs, it can be integrated tightly into existing OSS/BSS infrastructures to optimize work-flows.

1.5 Who should use the sip:provider CE?

The CE is specifically tailored to companies and engineers trying to start or experiment with a fully-featured SIP based VoIP service without having to go through the steep learning curve of SIP signalling, integrating the different building blocks to make them work together in a reasonable way and implementing the missing components to build a business on top of that.

In the past, creating a business-ready VoIP service included installation and configuration of SIP software like Asterisk, OpenSER, Kamailio etc., which can get quite difficult when it comes to implementing advanced features. It required to implement different web interfaces, billing engines and connectors to existing OSS/BSS infrastructure. These things are now obsolete due to the CE, which covers all these requirements.

2 Installation

2.1 Prerequisites

For *NGCP sip:provider CE*, it is mandatory that your production environment meets the following criteria:

HARDWARE REQUIREMENTS

- Recommended: Dual-core, x86_64 compatible, 3GHz, 4GB RAM, 128GB HDD
- Minimum: Single-core, x86_64 compatible, 1GHz, 512MB RAM, 16GB HDD

SUPPORTED OPERATING SYSTEMS

- Debian Lenny (5.0) 64-bit

INTERNET CONNECTION

- Hardware needs connection to the Internet

Important
Only **Debian Lenny (5.0) 64-bit** is currently supported as a host system for NGCP sip:provider CE.

Important
It is **HIGHLY** recommended that you use a **dedicated server** (either a physical or a virtual one) for sip:provider CE, because the installation process will wipe out existing MySQL databases and modify several system configurations.

2.2 Software Installation

2.2.1 Using the NGCP installer (recommended)

Installing the Operating System

You need to install Debian Lenny (5.0) 64-bit on the server. A **basic** installation without any additional task selection (like *Desktop System*, *Web Server* etc.) is sufficient.

Tip
Sipwise recommends using the Netinstall ISO as installation medium.

Important
If you use other kinds of installation media (e.g. provided by your hosting provider), prepare for some issues that might come up during installation. For example, you might be forced to manually resolve package dependencies in order to install the CE. Therefore, it is HIGHLY RECOMMENDED to use a clean Debian installation to simplify the installation process.

If you plan to install the CE on Virtual Hosting Providers like *Dreamhost* with their provided Debian installer, you might need to manually prepare the system for the NGCP installation, otherwise the installer will fail installing certain package versions required to function properly.

Using Dreamhost Virtual Private Server

A Dreamhost virtual server uses apt-pinning and installs specific versions of mysql and apache, so you need to clean this up beforehand.

```
apt-get remove --purge mysql-common ndn-apache22
mv /etc/apt/preferences /etc/apt/preferences.bak
apt-get update
apt-get dist-upgrade
```

Warning
Be aware that this step will break your web-based system administration provided by Dreamhost. Only do it if you are certain that you won't need it.

Installing the NGCP sip:provider CE

The *sip:provider CE* is based on the *Sipwise NGCP*, so download and install the *Sipwise NGCP* installer package:

```
PKG=ngcp-installer-latest.deb
wget http://deb.sipwise.com/spce/${PKG}
dpkg -i ${PKG}
```

Run the installer as root user:

```
ngcp-installer
```

The installer will ask you to confirm that you want to start the installation. Read the given information **carefully**, and if you agree, proceed with *y*.

The installation process will take a couple of minutes, depending on your network connection and server performance. If everything goes well, the installer will (depending on the language you use), show something like this:

```
Installation finished. Thanks for choosing NGCP sip:provider Community Edition.
```

During the installation, you can watch the background processing by executing the following command on a separate console:

```
tail -f /tmp/ngcp-installer.log
```

2.2.2 Using a preinstalled virtual machine

For quick test deployments, preinstalled virtualization images are provided. These images are intended to be used for quick test, not recommended for production use. At this moment, only virtualbox images are provided. We'll add more images compatible with the most popular virtualization solutions soon.

Virtualbox image

You can download a Virtualbox image from here. Once you have downloaded the file you can import it to Virtualbox via its import utility.

The format of the image is *ova*. If you have virtualbox 3.x running, which not compatible with *ova* format, you need to extract the file with any *tar* compatible software and import the *ovf* file which is inside the archive.

On Linux, you can do it like this:

```
tar -xvf sip_provider_2.x_virtualbox.ova
```

On Windows, right-click on the ova file, choose *Open with* and select *WinZIP* or *WinRAR* or any other application able to extract *tar* archives. Extract the files to any place and import the resulting *ovf* file in virtualbox.

Considerations when using this virtual machine:

- You will need a 64bit guest capable virtualbox setup.
- The root password is *sipwise*
- There's a user *sipwise* with password *sipwise*
- You should use *bridge mode* networking (adecuate your bridging iface in the virtual machine configuration) to avoid having the CE behind NAT.
- You'll need to adjust your timezone and keybord layout.
- The network configuration is set to dhcp. You'll need to change it to the apropiate static configuration.
- As the virtual image is a static file, it won't contain the most updated versions of our software. Please upgrade the system via apt as soon as you boot it for the first time.

2.3 Initial System Configuration

After the installation went through successfully, you are ready to adapt the system parameters to your needs to make the system work properly.

2.3.1 Network Configuration

The only parameter you need to change at this moment is the listening address for your SIP services. To do this, modify the parameter *networking→eaddress* in */etc/ngcp-config/config.yml*, which by default is set to *127.0.0.1*:

```
vim /etc/ngcp-config/config.yml
```

Look for the following section on top of the file:

```
networking:
  eaddress: 127.0.0.1

[...]
```

Change this parameter to the IP address configured during install time of the Debian operating system. If you haven't fully configured your network interfaces, do this by adapting also the file */etc/network/interfaces*:

```
vim /etc/network/interfaces
```

Add or adapt your interface configuration accordingly. For example, if you just want to use the system in your internal network 192.168.0.0/24, it could look something like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 1.2.3.4
        netmask 255.255.255.0
        gateway 1.2.3.1
        dns-nameservers 8.8.8.8
        dns-search yourdomain.com
```

```
/etc/init.d/networking restart
```

Warning
It is **HIGHLY** recommended that you use a public ip for the system. If you use a private ip, all your subscribers and gateways must be in the same private network. The CE handles gateways and subscribers behind NAT, but the CE itself cannot be behind NAT.

2.3.2 Apply Configuration Changes

In order to apply the changes you made to */etc/ngcp-config/config.yml*, you need to execute the following command to re-generate your configuration files and to automatically restart the services:

```
ngcpcfg apply
```

Tip
At this point, your system is ready to serve.

2.3.3 Securing your Server

During installation, the system user *cdrexport* is created. This jailed system account is supposed to be used to export CDR files via sftp/scp. Set a password for this user by executing the following command:

```
passwd cdrexport
```

The installer has set up a MySQL database on your server. You need to set a password for the MySQL root user to protect it from unauthorized access by executing this command:

```
mysqladmin password <your mysql root password>
```

For the *Administrative Web Panel* located at *https://<your-server-ip>:1443/*, a default user *administrator* with password *administrator* has been created. Connect to the panel (accept the SSL certificate for now) using this credentials and change the password of this user by going to *System Administration→Administrators* and clicking *edit*.

2.3.4 Configuring the Email Server

The NGCP installer will install *mailx* (which has *Exim4* as MTA as a default dependency) on the system, however the MTA is not configured by the installer. If you want to use the *Voicemail-to-Email* feature of the Voicebox, you need to configure your MTA properly. If you are fine to use the default MTA *Exim4*, execute the following command:

```
dpkg-reconfigure exim4-config
```

Depending on your mail setup in your environment (whether to use a smarthost or not), configure Exim accordingly. In the most simple setup, apply the following options when prompted for it:

- **General type of mail configuration:** `internet site; mail is sent and received directly using SMTP`
- **System mail name:** the FQDN of your server, e.g. `ce.yourdomain.com`
- **IP-addresses to listen on for incoming SMTP connections:** `127.0.0.1`
- **Other destinations for which mail is accepted:** the FQDN of your server, e.g. `ce.yourdomain.com`
- **Domains to relay mail for:** leave empty
- **Machines to relay mail for:** leave empty
- **Keep number of DNS-queries minimal (Dial-on-Demand)?** `No`
- **Delivery method for local mail:** `mbox format in /var/mail/`
- **Split configuration into small files?** `No`

Important
You are free to install and configure any other MTA (e.g. postfix) on the system, if you are more comfortable with that.

2.4 What's next?

To test and use your installation, you need to follow these steps now:

1. Create a SIP domain
2. Create some SIP subscribers

3. Register SIP endpoints to the system
4. Make local calls and test subscriber features
5. Establish a SIP peering to make PSTN calls

Please read the next chapter for instructions on how to do this.

3 Administrative Configuration

To be able to configure your first test clients, you will need a SIP domain and some subscribers in this domain. Throughout this steps, let's assume you're running the NGCP on the IP address *1.2.3.4*, and you want this IP to be used as SIP domain. This means that your subscribers will have an URI like *user1@1.2.3.4*.

Tip
You can of course set up a DNS name for your IP address (e.g. letting sip.yourdomain.com point to 1.2.3.4) and use this DNS name throughout the next steps, but we'll keep it simple and stick directly with the IP as a SIP domain for now.

Warning
Once you started adding subscribers to a SIP domain, and later decide to change the domain, e.g. from 1.2.3.4 to sip.yourdomain.com, you'll need to recreate all your subscribers in this new domain. It's currently not possible to easily change the domain part of a subscriber.

Go to the *Administrative Web Panel* (*Admin Panel*) running on *https://<ce-ip>:1443/* and follow the steps below. The default user on the system is *administrator* with the password *administrator*, if you haven't changed it already in [?simpara].

3.1 Creating Domains

Go to *System Administration→Domains*. You'll see a form *Create Domain*, where you have to provide the name of your SIP domain. In our example we'll put in *1.2.3.4* there and click *add*. The newly created domain now shows up under *Edit Domains*.

3.1.1 Defining Domain Dialplans

Important
On the NGCP, every phone number is treated in E.164 format *<country code><area code><subscriber number>*. With domain dialplans, you can set regular expressions as rewrite rules to normalize user-provided numbers to the required format.

Within every of your SIP domains, you can define rewrite rules for the caller and the callee. This is used to define what an end user can dial for outbound calls, and what is displayed as the calling party on inbound calls. To define these rules, click on your newly created domain.

Tip
In Europe, the following formats are widely accepted: *+<cc><ac><sn>*, *00<cc><ac><sn>* and *0<ac><sn>*. Within this section, we will use these formats to show how to use rewrite rules to normalize and denormalize number formats.

Inbound Rewrite Rules for Caller

These rules are used to normalize user-provided numbers (e.g. passed in *From Display Name* or *P-Preferred-Identity* headers) into E.164 format. In our example, we'll normalize the three different formats mentioned above into E.164 format.

STRIP LEADING `00` OR `+`

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`
- Priority: `50`
- Description: `International to E.164`

REPLACE `0` BY AUSTRIAN COUNTRY CODE `43`:

- Match Pattern: `^0([1-9][0-9]+)$`
- Replacement Pattern: `43\1`
- Priority: `50`
- Description: `Austria National to E.164`

Tip
When routing a call, the rewrite processing is stopped after the first match of a rule. If you have two rules (e.g. a generic one and a more specific one), where both of them would match some numbers, set the *Priority* value of the specific rule lower than for the generic rule, so the specific one is tried first. The lower the value, the more precedence it gets.

Inbound Rewrite Rules for Callee

These rules are used for the number the end user dials to place a call. In our example, we again allow the three different formats mentioned above, so we put in the same rules as for the caller.

STRIP LEADING `00` OR `+`

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`
- Priority: `50`
- Description: `International to E.164`

REPLACE `0` BY AUSTRIAN COUNTRY CODE `43`:

- Match Pattern: `^0([1-9][0-9]+)$`
- Replacement Pattern: `43\1`
- Priority: `50`
- Description: `Austria National to E.164`

Tip
Our provided rules will only match if the caller dials a numeric number. If he dials an alphanumeric SIP URI, none of our rules will match and no rewriting will be done. You can however define rules for that as well. For example, you could allow your end users to dial `support` and rewrite that to your support hotline using the match pattern `^support$` and the replace pattern `43800999000` or whatever your support hotline number is.

Outbound Rewrite Rules for Caller

These rules are used to rewrite the calling party number for a call to an end user. For example, if you want the device of your end user to show *0<ac><sn>* if a national number calls this user, and *00<cc><ac><sn>* if an international number calls, put the following rules there.

REPLACE AUSTRIAN COUNTRY CODE `43` BY `0`

- Match Pattern: `^43([1-9][0-9]+)$`
- Replacement Pattern: `0\1`
- Priority: `50`
- Description: `E.164 to Austria National`

PREFIX `00` FOR INTERNATIONAL CALLER

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `00\1`
- Priority: `51`
- Description: `E.164 to International`

Tip
Note that both of the rules would match a number starting with `43`, so we raise the Priority value in order to try matching the national rule first.

3.2 Creating Accounts

An *account* on the NGCP is a billing container, which contains one or more subscribers for a customer. In this billing container, you can define which *Billing Profile* is used for calls being placed by the subscribers of this account.

To create a new account, go to *User Administration→Accounts* and click *Create new account*. For our first tests, we will use the default values, so just click *Save*.

You will be presented with an overview of the new account, showing basic *Account Information*, the *Account Balance* (which will only get relevant when you start using your own Billing Profile) and the list of *Subscribers* for this account, which is currently empty.

3.3 Creating Subscribers

In the *Subscribers* section at the bottom of the account information for the account you created before, click *create new* to create a new subscriber for this account. You will be presented with the *Master Data* form, where you have to fill in the following options:

- **web username**: This is the user part of the username the subscriber may use to log into her *Customer Self Care Interface*. The user part will be automatically suffixed by the SIP domain you choose for the **SIP URI**. Usually the web username is identical to the **SIP URI**, but you may choose a different naming schema.

Caution
The web username needs to be unique. The system will return a fault if you try to use the same web username twice.

- **web password**: This is the password for the subscriber to log into her *Customer Self Care Interface*. It must be at least 6 characters long.
- **E.164 number**: This is the telephone number mapped to the subscriber, separated into *Country Code (CC)*, *Area Code (AC)* and *Subscriber Number (SN)*. For the first tests, you can set a made-up number here and change it later when you get number blocks assigned by your PSTN interconnect partner. So in our example, we'll use *43* as CC, *99* as AC and *1001* as SN to form the phantasy number *+43 99 1001*.

Tip
This number can actually be used to place calls between local subscribers, even if you don't have any PSTN interconnection. This comes in handy if you use phones instead of soft-clients for your tests. The format in which this number can be dialled so the subscriber is reached has been defined previously in Section 3.1.1.

- **SIP URI**: Insert the user part of the URI into the first field (e.g. *user1*) and select the domain you want to put this subscriber into in the drop-down.

Caution
With the default system settings, the user part has to have at least one alphabetic character, so it's not possible by default to just use a number here. To allow that, on the console set *ossbss→provisioning→allow_numeric_usernames* to *1* in */etc/ngcp-config/config.yml* and execute the command `ngcpcfg apply`. If you want for example to set a numeric customer id as the SIP user, make sure it does not overlap with actual phone numbers, otherwise these calls will be routed to the SIP user instead of the phone number.

- **SIP password**: The password of your subscriber to authenticate on the SIP proxy. It must be at least 6 characters long.
- **administrative**: If you have multiple subscribers in one account and set this option for one of them, this subscriber can administrate other subscribers via the *Customer Self Care Interface*.

Click *Save* to create the subscriber. Repeat the creation of accounts and subscribers for all your test accounts. You should have at least 3 subscribers to test all the functionality of the NGCP.

Tip
At this point, you're able to register your subscribers to the NGCP and place calls between these subscribers.

3.4 Creating Peerings

If you want to terminate calls at or allow calls from 3rd party systems (e.g. PSTN gateways, SIP trunks), you need to create SIP peerings for that. To do so, go to *System Administration→SIP Peerings*. There you can add peering groups, and for each peering group add peering servers. Every peering group needs a peering contract for correct interconnection billing.

3.4.1 Creating Peering Contracts

In order to create peering groups, you must create at least one peering contract. It defines, which billing profile is going to be used for calls to the corresponding peering groups. In this example, we will use the *Default Billing Profile*, because we haven't set up proper profiles yet.

In the *SIP Peering Contracts* section, click *create new* to add a peering contract. You will be presented with a form to set the billing profile and some contact details for the contract. To create a very basic dummy profile, we will set the following values:

- **billing profile:** `Default Billing Profile`

- **First Name:** leave empty
- **Last Name:** leave empty
- **Company:** leave empty

Click *Save* to store the peering contract.

3.4.2 Creating Peering Groups

In *System Administration→SIP Peerings*, create a new peering group in the section *Create Peering Group*. You will usually have one peering group per carrier you're planning to send traffic to and receive traffic from. We will create a test group using the peering contract added before:

- **Name:** `test group`
- **Priority:** `1`
- **Description:** `peering to a test carrier`
- **Peering Contract:** select the id of the contract created before

Then click *add* to create the group.

3.4.3 Creating Peering Servers

In the group created before, you need to add peering servers to route calls to and receive calls from. To do so, click on the *Name* of your created group in the section *SIP Peering Groups*.

Then add your first peering server in the section *Peering Servers*. In this example, we will create a peering server with IP *2.3.4.5* and port *5060*:

- **Name**: `test-gw-1`
- **IP Address:** `2.3.4.5`
- **Port:** `5060`
- **Weight:** `1`

Then click *add* to create the peering server.

You will now see an additional section *Peering Rules* after your list of peering servers. There you have to define which numbers to route via this peering group.

Important
If you do not add at least one peering rule to your group, the servers in this group will NOT be used for outbound calls. The NGCP will however allow inbound calls from the servers in this group even without peering rules.

Since the previously created peering group will be the only one in our example, we have to add a default rule to route *all* calls via this group. To do so, create a new peering rule with the following values:

- **Callee Prefix:** leave empty
- **Caller Pattern:** leave empty
- **Description:** `Default Rule`

Then click *add* to add the rule to your group.

Important
The selection of peering servers for outbound calls is done in the following order: **1.** Length of the matching peering rules for a call. **2.** Priority of the peering group. **3.** Weight of the peering servers in the selected peering group. After one or more peering group is matched for an outbound call, all servers in this group are tried, according to their weight (lower weight has more precedence). If a peering server replies with SIP codes `408`, `500` or `503`, or if a peering server doesn't respond at all, the next peering server in the current peering group is used as a fallback, one after the other until the call succeeds. If no more servers are left in the current peering group, the next group which matches the peering rules is going to be used.

3.4.4 Creating Dialplans for Peering Servers

For each peering server, you can define rewrite rules similar to the rewrite rules in Section 3.1.1.

If your peering servers don't send numbers in E.164 format <*cc*><*ac*><*sn*>, you need to create *Inbound Rewrite Rules* for each peering server to normalize the numbers for caller and callee to this format, e.g. by stripping leading + or put them from national into E.164 format.

Likewise, if your peering servers don't accept this format, you need to create *Outbound Rewrite Rules* for each of them, for example to append a + to the numbers.

4 Customer Self-Care Configuration

There are two ways for end users to maintain their subscriber settings: via the *Customer Self-Care Web Interface* and via *Vertical Service Codes* using their SIP phones.

4.1 The Customer Self-Care Web Interface

The NGCP provides a web panel for end users (CSC panel) to maintain their subscriber accounts, which is running on *https://<ce-ip>*. Every subscriber can log in there, change subscriber feature settings, view their call lists and trigger calls using the click-to-dial feature.

4.1.1 Login Procedure

To log into the CSC panel, the end user has to provide his full web username (e.g. `user1@1.2.3.4`) and the web password defined in Section 3.3. Once logged in, he can change his web password in the *Account* section. This will NOT change his SIP password, so if you control the end user devices, you can auto-provision the SIP password into the device and keep it secret, and just hand over the web password to the customer. This way, the end user will only be able to place calls with this auto-provisioned device and not with an arbitrary soft-phone, but can nonetheless manage his account via the CSC panel.

Important
You can simplify the login procedure for one SIP domain in such a way that users in this domain only need to pass the user part (e.g. `user1`) as a username instead of the full web username to log in by setting the parameter *www_csc→site_domain* in the config file */etc/ngcp-config/config.yml* to the corresponding domain (e.g. `1.2.3.4`) and execute `ngcpcfg apply`.

4.1.2 Site Customization

As an operator, you can change the appearance of the CSC panel by modifying a couple of parameters in the section *www_csc→site_confi* of the config file */etc/ngcp-config/config.yml*. Modify the site title, your company details and the logo to reflect your use case.

You can also enable/disable specific languages a user can choose from in the CSC panel. Currently, English (`en`) and Spanish (`es`) are supported and are activated by default.

After changing one or more of the parameters in this file, execute `ngcpcfg apply` to activate the changes.

4.2 The Vertical Service Code Interface

Vertical Service Codes (VSC) are codes a user can dial on his phone to provision specific features for his subscriber account. The format is `*<code>*<value>` to activate a specific feature, and `#<code>` or `#<code>#` to deactivate it. The *code* parameter is a two-digit code, e.g. `72`. The *value* parameter is the value being set for the corresponding feature.

Important
The *value* user input is normalized using the domain rewrite rules defined in Section 3.1.1.

By default, the following codes are configured for setting features. The examples below assume that there is a domain rewrite rule normalizing the number format *0<ac><sn>* to *<cc><ac><sn>* using *43* as country code.

- **72** - enable *Call Forward Unconditional* e.g. to 431000 by dialing `*72*01000`, and disable it by dialing `#72`.
- **90** - enable *Call Forward on Busy* e.g. to 431000 by dialing `*90*01000`, and disable it by dialing `#90`.
- **92** - enable *Call Forward on Timeout* e.g. after 30 seconds of ringing to 431000 by dialing `*92*30*01000`, and disable it by dialing `#92`.
- **93** - enable *Call Forward on Not Available* e.g. to 431000 by dialing `*93*01000`, and disable it by dialing `#93`.
- **50** - set *Speed Dial Slot*, e.g. set slot 1 to 431000 by dialing `*50101000`, which then can be used by dialing `*1`.
- **55** - set *One-Shot Reminder Call* e.g. to 08:30 by dialing `*55*0830`.

You can change any of the codes (but not the format) in */etc/ngcp-config/config.yml* in the section *sems→vsc*. After the changes, execute `ngcpcfg apply`.

Important
Note that you can also change the parameter *voicemail_number* in this section of the config file. If you use a Call Forward VSC with this number, the forward is set to the Voicebox. E.g. when setting this parameter to *9999*, a Call Forward Unconditional to the Voicebox is set if the user dials `*72*9999`.

Caution
If you have the EMTAs under your control, make sure that the specified VSCs don't overlap with EMTA-internal VSCs, because the VSC calls must be sent to the NGCP via SIP like normal telephone calls.

5 Billing Configuration

This chapter describes the steps necessary to rate calls and export rated CDRs (call detail records) to external systems.

5.1 Billing Data Import

Service billing on the NGCP is based on billing profiles, which may be assigned to VoIP accounts and SIP peerings. The design focuses on a simple, yet flexible approach, to support arbitrary dial-plans without introducing administrative overhead for the system administrators. The billing profiles may define a base fee and free time or free money per billing interval. Unused free time or money automatically expires at the end of the billing interval.

Each profile may have call destinations (usually based on E.164 number prefix matching) with configurable fees attached. Call destination fees each support individual intervals and rates, with a different duration and/or rate for the first interval. (e.g.: charge the first minute when the call is opened, then every 30 seconds, or make it independent of the duration at all) It is also possible to specify different durations and/or rates for peak and off-peak hours. Peak time may be specified based on weekdays, with additional support for manually managed dates based on calendar days. The call destinations can finally be grouped for an overview on user's invoices by specifying a zone in two detail levels. (E.g.: national landline, national mobile, foreign 1, foreign 2, etc.)

5.1.1 Creating Billing Profiles

The first step when setting up billing data is to create a billing profile, which will be the container for all other billing related data. Go to *System Administration→Billing* and click on *create new billing profile*. You will be taken to a web form where you may enter the following parameters (all values except *handle* and *name* may be left empty):

- **handle**: A unique, permanently fixed string which is used to attach the billing profile to a VoIP account or SIP peering contract.
- **name**: A free form string used to identify the billing profile in the *Admin Panel*. This may be changed at any time.
- **interval charge**: A base fee for the billing interval, specifying a monetary amount (represented as a floating point number) in whatever currency you want to use.
- **interval free time**: If you want to include free calling time in your billing profile, you may specify the number of seconds that are available every billing interval. See *Creating Billing Fees* below on how to select destinations which may be called using the free time.
- **interval free cash**: Same as for *interval free time* above, but specifies a monetary amount which may be spent on outgoing calls. This may be used for example to implement a minimum turnover for a contract, by setting the *interval charge* and *interval free cash* to the same values.
- **currency**: The currency symbol for your currency. Any UTF-8 character may be used and will be printed in web interfaces.
- **VAT rate**: The percentage of value added tax for all fees in the billing profile. Currently for informational purpose only and not used further.
- **VAT included**: Whether VAT is included in the fees entered in web forms or uploaded to the platform. Currently for informational purpose only and not used further.

5.1.2 Creating Billing Fees

To set up billing fees, go to *System Administration→Billing* and select *edit fees* next to the billing profile you want to configure. Billing fees may be uploaded using a configurable CSV file format, or entered directly via the web interface by selecting *create new entry* just below *Stored Billing Fees*. To configure the CSV field order for the file upload, rearrange the entries in the *www_admin→fees_csv→element_order* array in */etc/ngcp-config/config.yml* and execute the command `ngcpcfg apply`. For input via the web interface, just fill in the text fields accordingly. In both cases, the following information may be specified independently for every destination:

- **destination**: The destination E.164 prefix, SIP domain (or IP address) or SIP URI. May be a simple string (e.g. `431`, `sip.sipwise.com` or `someone@sip.sipwise.com`) or a regular expression matching the complete E.164 number, SIP domain or SIP URI (e.g. `^431.*$`, `^.*@sip\.sipwise\.com$` or `^someone@sip\.sipwise\.com$`). Regular expressions will be stored unmodified, plain strings will be extended exactly as shown in the two examples. The web interface will remove the regular expression prefix and suffix from an entry in the list of store billing fees.

Important
The destination needs to be unique for a billing profile. The system will return an error if a destination is specified twice, both for the file upload and the input via the web interface.

- **zone**: A zone name for a group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. `foreign zone 1`)
- **zone detail**: A zone name for a more detailed group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. `germany landline`)
- **onpeak init rate**: The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours.
- **onpeak init interval**: The duration of the first billing interval, in seconds. Applicable to calls during onpeak hours.
- **onpeak follow rate**: The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours. Defaults to *onpeak init rate*.
- **onpeak follow interval**: The duration of subsequent billing intervals, in seconds. Applicable to calls during onpeak hours. Defaults to *onpeak init interval*.
- **offpeak init rate**: The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *onpeak init rate*.
- **offpeak init interval**: The duration of the first billing interval, in seconds. Applicable to calls during off-peak hours. Defaults to *onpeak init interval*.
- **offpeak follow rate**: The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *offpeak init rate* if that one is specified, or to *onpeak follow rate* otherwise.
- **offpeak follow interval**: The duration of subsequent billing intervals, in seconds. Applicable to calls during off-peak hours. Defaults to *offpeak init interval* if that one is specified, or to *onpeak follow interval* otherwise.
- **use free time**: Specifies whether free time minutes may be used when calling this destination. May be specified in the file upload as 0, n[o], f[alse] and 1, y[es], t[rue] respectively.

5.1.3 Creating Off-Peak Times

To be able to differentiate between on-peak and off-peak calls, the platform stores off-peak times for every billing profile based on weekdays and/or calendar days. To edit the settings for a billing profile, go to *System Administration→Billing* and select *edit peak times* next to the billing profile you want to configure.

To set off-peak times for a weekday, click on *edit* next to the according weekday. You will be presented with two input fields which both receive a timestamp in the form of *hh:mm:ss* specifying a time of day for the start and end of the off-peak period. If any of the fields is left empty, the system will automatically insert `00:00:00` (*start* field) or *23:59:59* (*end* field). Click on *save* to store the setting in the database. You may create more than one off-peak period per weekday, and you may edit existing entries using any of the input fields and clicking *save* next to it. To completely delete a range, just select *delete* next to the entry.

To specify off-peak ranges based on calendar dates, click on *add new date* just below *Dates*. Enter a date in the form of *YYYY-MM-DD* into the *date* input field and fill in the *start* and *end* timestamps as outlined above. Select *save* to store the entry, or *cancel* to close the input form. Existing dates will be listed below, grouped by year. Click on any of the years to view all dates which have been recorded for it. If an entry is added, the corresponding year is expanded automatically. If an already existing date is added, it will overwrite the existing entry.

5.2 Billing Data Export

Regular billing data export is done using CSV (*comma separated values*) files which may be downloaded from the platform using the *cdrexport* user which has been created during the installation.

5.2.1 File Name Format

In order to be able to easily identify billing files, the file name is constructed by the following fixed-length fields:

```
<prefix><separator><version><separator><timestamp><separator><sequence number><suffix>
```

The definition of the specific fields is as follows:

Table 1: CDR export file name format

File name element	Length	Description
`<prefix>`	7	A fixed string. Always `sipwise`.
`<separator>`	1	A fixed character. Always _.
`<version>`	3	The format version. Always `001`.
`<timestamp>`	14	The file creation timestamp in the format `YYYYMMDDhhmmss`.
`<sequence number>`	10	A unique 10-digit zero-padded sequence number for quick identification.
`<suffix>`	4	A fixed string. Always `.cdr`.

A valid example filename for a billing file created at 2007-11-10 12:30:00 and being the fourth file exported by the system, is:

```
sipwise_001_20071110123000_0000000004.cdr
```

5.2.2 File Format

Each billing file consists of three parts: one header line, zero to fivethousand body lines and one trailer line.

File Header Format

The billing file header is one single line, which is constructed by the following fields:

```
<number of records>
```

The `<number of records>` is a 4-digit integer number, specifying the number of body lines contained in the file. A valid example for a Header is:

```
0738
```

File Body Format

The body consists of a minimum of zero and a maximum of 5000 lines. Each line holds one call detail record in CSV format and is constructed by the following fields, all of them enclosed in single quotes:

Table 2: CDR export file body line format

Body Element	Length	Type	Description
`<id>`	1-10	uint	Internal CDR id.
`<update_time>`	19	timestamp	Timestamp of last modification.
`<source_user_id>`	36	string	Internal UUID of calling party.
`<source_provider_id>`	1-255	string	Internal ID of calling party provider.

Table 2: (continued)

Body Element	Length	Type	Description
<source_user>	1-255	string	SIP username of calling party.
<source_domain>	1-255	string	SIP domain of calling party.
<source_cli>	1-64	string	CLI of calling party in E.164 format.
<source_clir>	1	uint	1 for calls with CLIR, 0 otherwise
<destination_user_id>	1 / 36	string	Internal UUID of called party or 0 if callee is not local
<destination_provider_id>	1-255	string	Internal ID of called party provider.
<destination_user>	1-255	string	Final SIP username of called party.
<destination_domain>	1-255	string	Final SIP domain of called party.
<destination_user_in>	1-255	string	Incoming SIP username of called party.
<destination_domain_in>	1-255	string	Incoming SIP domain of called party.
<call_type>	3-4	string	The type of the call - one of: call: normal call cfu: call forward unconditional cft: call forward timeout cfb: call forward busy cfna: call forward no answer
<call_status>	2-7	string	The final call status - one of: ok: successful call busy: callee busy noanswer: no answer from callee cancel: cancel from caller offline callee offline timeout: no reply from callee other: unspecified, see <call_code> for details
<call_code>	3	uint	The final SIP status code.
<start_time>	19	timestamp	Timestamp of call start.
<duration>	10	uint	Length of call in seconds
<call_id>	1-255	string	The SIP call-id.
<rating_status>	2-7	string	The internal rating status - one of: unrated: not rated ok: successfully rated failed: error while rating Currently always ok or unrated, depending on whether rating is enabled or not.
<rated_at>	0 / 19	timestamp	Timestamp of rating or empty if not rated.
<carrier_cost>	4-11	fixed precision	The carrier termination cost or empty if not rated. In cent with two decimals.
<customer_cost>	4-11	fixed precision	The customer cost or empty if not rated. In cent with two decimals.
<carrier_zone>	0-127	string	The carrier billing zone or empty if not rated.
<customer_zone>	0-127	string	The customer billing zone or empty if not rated.
<carrier_destination>	0-127	string	The carrier billing destination or empty if not rated.
<customer_destination>	0-127	string	The customer billing destination or empty if not rated.
<dialed_digits>	1-255	string	The user-part of the SIP Request URI as received by the soft-switch.
<line_terminator>	1	string	A fixed character. Always \n (special char LF – ASCII 0x0A)

A valid example of one body line of a rated CDR is (linebrakes added for display):

```
'2055','2007-11-07 11:36:49','6b6977f9-6125-4339-a82c-3c5af04652d2', '1','test',
```

```
'sipwise.com','43720456700','0','0','3','4315551234','192.168.101.17','4315551234',
'sipwise.com','call','ok','200','2007-11-05 16:17:37','74',
'7F2A3EA1-472F34108EE84@192.168.101.11','ok','2007-11-07 11:36:49','9.25','16.03',
'national landline','national landline','landline vienna','landline vienna','015551234'
```

File Trailer Format

The billing file trailer is one single line, which is constructed by the following fields:

```
<md5 sum>
```

The `<md5 sum>` is a 32 character hexadecimal MD5 hash of the *Header* and *Body*.

To validate the billing file, one must remove the Trailer before computing the MD5 sum of the file. An example bash script to validate the integrity of the file is given below:

```
#!/bin/sh

error() { echo $@; exit 1; }
test -n "$1" || error "Usage: $0 <cdr-file>"
test -f "$1" || error "File '$1' not found"

TMPFILE="/tmp/$(basename "$1")"
MD5="$(sed -rn '$ s/^([a-z0-9]{32}).*$/\1/i p' "$1")  $TMPFILE"
sed '$d' "$1" > "$TMPFILE"
echo "$MD5" | md5sum -c -
rm -f "$TMPFILE"
```

Given the script is located in `cdr-md5.sh` and the CDR-file is `sipwise_001_20071110123000_0000000004.cdr`, the output of the integrity check for an intact CDR file would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: OK
```

If the file has been altered during transmission, the output of the integrity check would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: FAILED
md5sum: WARNING: 1 of 1 computed checksum did NOT match
```

5.2.3 File Transfer

Billing files are created twice per hour at minutes `25` and `55` and are stored in the home directory of the `cdrexport` user. If the amount of records within the transmission interval exceeds the threshold of 5000 records per file, multiple billing files are created. If no billing records are found for an interval, a billing file without body data is constructed for easy detection of lost billing files on the 3rd party side.

CDR files are fetched by a 3rd party billing system using SFTP or SCP with either public key or password authentication using the username cdrexport. If public key authentication is chosen, the public key file has to be stored in the file `~/.ssh/authorized_keys` below the home directory of the `cdrexport` user. Otherwise, a password has to be set for the user.

The 3rd party billing system is responsible for deleting CDR files after fetching them.

Note
The `cdrexport` user is kept in a jailed environment on the system, so it has only access to a very limited set of commandline utilities.

6 Configuration Framework

The CE provides a configuration framework for consistent and easy to use low level settings management. A basic usage of the configuration framework only needs two actions already used in previous chapters:

- Edit */etc/ngcp-config/config.yml* file.
- Execute *ngcpcfg apply* command.

Low level management of the configuration framework might be required by advanced users though. This chapter explains the architecture and usage of the NGCP configuration framework. If the basic usage explained above fits your needs, feel free to skip this chapter and return to it when your requirements change.

A more detailed workflow of the configuration framework for creating a configuration file consists of 6 steps:

- Generation or editing of configuration templates and/or configuration values.
- Generation of the configuration files based on configuration templates and configuration values defined in config.yml and constants.yml files.
- Execution of *prebuild* commands if defined for a particular configuration file or configuration directory.
- Placement of the generated configuration file in the target directory. This step is called *build* in the configuration framework.
- Execution of *postbuild* commands if defined for that configuration file or configuration directory.
- Execution of *services* commands if defined for that configuration file or configuration directory. This step is called *services* in the configuration framework.
- Saving of the generated changes. This step is called *commit* in the configuration framework.

6.1 Configuration templates

The CE provides configuration file templates for most of the services it runs. These templates are stored in the directory */etc/ngcp-config/templates*.

Example: Template files for */etc/sems/sems.conf* are stored in */etc/ngcp-config/templates/etc/sems/*.

There are different types of files in this template framework, which are described below.

6.1.1 .tt2 and .customtt.tt2 files

These files are the main template files that will be used to generate the final configuration file for the running service. They contain all the configuration options needed for a running CE system. The configuration framework will combine these files with the values provided by *config.yml* and *constants.yml* to generate the appropriate configuration file.

Example: In the installation chapter we've changed the parameter *networking → eaddress* from the default *127.0.0.1* to our public IP address *1.2.3.4*. This parameter will for example change kamailio's listen address, when the configuration file is generated. A quick look to the template file under */etc/ngcp-config/templates/etc/kamailio/kamailio.cfg.tt2* will show a line like this:

```
listen=udp:[% networking.eaddress %]:[% kamailio.port %]
```

After applying the changes with the *ngcpcfg apply* command, a new configuration file will be created under */etc/kamailio/kamailio.cfg* with the proper values taken from the main configuration file:

```
listen=udp:1.2.3.4:5060
```

All the low-level configuration is provided by these .tt2 template files and the corresponding config.yml file. Anyways, advanced users might require a more particular configuration.

Instead of editing .tt2 files, the configuration framework recognises .customtt.tt2 files. These files are the same as .tt2, but they have higher priority when the configuration framework creates the final configuration files. An advanced user should create a .customtt.tt2 file from a copy of the corresponding .tt2 template and leave the .tt2 template untouched. This way, the user will have his personalized configuration and the system will continue providing a working, updated configuration template in .tt2 format.

Example: We'll create */etc/ngcp-config/templates/etc/kamailio.cfg.customtt.tt2* and use it for our personalized configuration. In this example, we'll just append a comment at the end of the template.

```
cd /etc/ngcp-config/templates/etc/kamailio
cp kamailio.cfg.tt2 kamailio.cfg.customtt.tt2
echo '# This is my last line comment' >> kamailio.cfg.customtt.tt2
ngcpcfg apply
```

The ngcpcfg command will generate */etc/kamailio/kamailio.cfg* from our custom template instead of the general one.

```
tail -1 /etc/kamailio/kamailio.cfg
# This is my last line comment
```

Tip
The tt2 files use the Template Toolkit language. Therefore you can use all the feature this excellent toolkit provides within ngcpcfg's template files (all the ones with the .tt2 suffix).

6.1.2 .prebuild and .postbuild files

After creating the configuration files, the configuration framework can execute some commands before and after placing that file in its target directory. These commands usually are used for changing the file's owner, groups, or any other attributes. There are some rules these commands need to match:

- They have to be placed in a *.prebuild* or *.postbuild* file in the same path as the original *.tt2* file.
- The file name must be the same as the configuration file, but having the mentioned suffixes.
- The commands must be *bash* compatible.
- The commands must return 0 if successful.
- The target configuration file is matched by the environment variable *output_file*.

Example: We need *www-data* as owner of the configuration file */etc/ngcp-ossbss/provisioning.conf*. The configuration framework will by default create the configuration files with root:root as owner:group and with the same permissions (rwx) as the original template. For this particular example, we will change the owner of the generated file using the *.postbuild* mechanism.

```
echo 'chgrp www-data ${output_file}' \
  > /etc/ngcp-config/templates/etc/ngcp-ossbss/provisioning.conf.postbuild
```

6.1.3 .services files

.services files are pretty similar and might contain commands that will be executed after the *build* process. There are two types of *.services* files:

- The particular one, with the same name as the configuration file it is associated to. Example: */etc/ngcp-config/templates/etc/asterisk/sip* is associated to */etc/asterisk/sip.conf*

- The general one, named *ngcpcfg.services* wich is associated to every file in its target directory. Example: */etc/ngcp-config/templates/etc* is associated to every file under */etc/asterisk/*

When the *services* step is triggered all *.services* files associated to a changed configuration file will be executed. In case of the general file, any change to any of the configuration files in the directory will trigger the execution of the commands.

Tip
If the service script has the execute flags set (*chmod +x $file*) it will be invoked directly. If it doesn't have execute flags set it will be invoked under bash. Make sure the script is bash compatible if you do not set execute permissions on the service file.

These commands are usually service reload/restarts to ensure the new configuration has been loaded by running services.

Note
The configuration files mentioned in the following example usually already exist on the platform. Please make sure you don't overwrite any existing files if following this example.

Example:

```
echo '/etc/init.d/mysql restart' \
  > /etc/ngcpcfg-config/templates/etc/mysql/my.cnf.services
echo '/etc/init.d/asterisk restart' \
  > /etc/ngcpcfg-config/templates/etc/asterisk/ngcpcfg.services
```

In this example we created two *.services* files. Now, each time we trigger a change to */etc/mysql.my.cnf* or to */etc/asterisk/** we'll see that MySQL or Asterisk services will be restarted by the ngcpcfg system.

6.2 config.yml and constants.yml files

The */etc/ngcp-config/config.yml* file contains all the user-configurable options, using the YAML (YAML Ain't Markup Language) syntax.

The */etc/ngcp-config/constants.yml* file provides configuration options for the platform that aren't supposed to be edited by the user. Do not manually edit this file unless you really know what you're doing.

The */etc/ngcp-config/ngcpcfg.cfg* file is the main configuration file for ngcpcfg itself. Do not manually edit this file unless you really know what you're doing.

6.3 ngcpcfg and its command line options

The ngcpcfg utility supports the following command line options:

6.3.1 apply

The *apply* option is a short-cut for the options "build && services && commit" and also executes *etckeeper* to record any modified files inside /etc. It is the recommended option to use the ngcpcfg framework unless you want to execute any specific commands as documented below.

6.3.2 build

The *build* option generates (and therefore also updates) configuration files based on their configuration (config.yml) and template files (.tt2). Before the configuration file is generated a present .prebuild will be executed, after generation of the configuration file the according .postbuild script (if present) will be executed. If a *file* or *directory* is specified as argument the build will generate only the specified configuration file/directory instead of running through all present templates.

Example: to generate only the file /etc/apache2/sites-available/ngcp-www-admin you can execute:

```
ngcpcfg build /etc/apache2/sites-available/ngcp-www-admin
```

Example: to generate all the files located inside the directory /etc/apache2/ you can execute:

```
ngcpcfg build /etc/apache2/
```

6.3.3 commit

The *commit* option records any changes done to the configuration tree inside */etc/ngcp-config*. The commit option should be executed when you've modified anything inside the configuration tree.

6.3.4 help

The *help* options displays ngcpcfg's help screen and then exits without any further actions.

6.3.5 initialise

The *initialise* option sets up the ngcpcfg framework. This option is automatically executed by the installer for you, so you shouldn't have to use this option in normal operations mode.

6.3.6 services

The *services* option executes the service handlers for any modified configuration file(s)/directory.

6.3.7 status

The *status* option provides a human readable interface to check the state of the configuration tree. If you are unsure what should be done as next step or if want to check the current state of the configuration tree just invoke *ngcpcfg status*.

If everything is OK and nothing needs to be done the output should look like:

```
# ngcpcfg status
Checking state of ngcpcfg:
OK:   has been initialised already (without shared storage)
Checking state of configuration files:
OK:   nothing to commit.
Checking state of /etc files
OK:   nothing to commit.
```

If the output doesn't say "OK" just follow the instructions provided by the output of *ngcpcfg status*.

7 Provisioning interfaces

The CE provides two provisioning interfaces for easy interconnection with 3rd party tools. The user can access all the functionalities provided by the Admin interface or the CSC interface via SOAP or XMLRPC interfaces. The server provides online documentation about all the functions available. To access the online documentation for the first time, you need to follow the following instructions:

- Generate a password for http access to the provisioning interfaces:

```
htpasswd -nbs mypassword myuser
```

- Edit */etc/ngcp-config/config.yml*. Under section *ossbss→htpasswd*, replace *user* and *pass* with your new values and execute *ngcpcfg apply* as usual.
- Access https://<ce-ip>:2443/SOAP/Provisioning.wsdl and login with your new credentials.

Note
The default port for provisioning interfaces is 2443. You can change it in */etc/ngcp-config/config.yml* by modifying *ossbss→apache→port* and execute *ngcpcfg apply*.

Important
The displayed online API documentation shows all the currently available functionalities. Enabling or disabling features in */etc/ngcp-config/config.yml* will directly reflect in the functions being available via the APIs.

8 Security and Maintenance

Once the CE is in production, security and maintenance becomes really important. In this chapter, we'll go through a set of best practices for any production system.

8.1 Firewalling

The CE runs a wide range of services. Some of them need to interact with the user, while some others need to interact with the administrator or with nobody at all. Assuming that we trust the CE server for outgoing connections, we'll focus only on incoming traffic to define the services that need to be open for interaction.

Table 3: Subscribers

Service	Default port	Config option
Customer self care interface	443 TCP	www_csc→apache→port
SIP	5060 UDP	kamailio→port
SIP	5060 TCP	kamailio→port + kamailio→disable_tcp (Disabled by default)
RTP	30000-40000 UDP	rtpproxy→minport + rtpproxy→maxport

Table 4: Administrators

Service	Default port	Config option
SSH/SFTP	22 TCP	NA
Administrator interface	1443 TCP	www_admin→apache→port
Provisioning interfaces	2443 TCP	ossbss→apache→port

8.2 Password management

The CE comes with some default passwords the user should change during the deployment of the system. They have been explained in the previous chapters of this document.

- The default password of the system account *cdrexport* is *cdrexport*. Although this is a jailed account, it has access to sensitive information, namely the Call Detail Records of all calls. SSH keys should be used to login this user, or alternatively a really strong password should be generated.
- The *root* user in MySQL has no default password. A password should be set using the *mysqladmin password* command.
- The administrative web interface has a default user *administrator* with password *administrator*. It should be changed within this interface.

Important
Many NGCP services use MySQL backend. Users and passwords for these services are created during the installation. These passwords are unique for each installation, and the connections are restricted to localhost. You should not change these users and passwords.

8.3 SSL certificates.

The CE provides default, self-signed SSL certificates for SSL connections. These certificates are common for every installation. Before going to production state, the system administrator should provide SSL certificates for the web services. These certificates can either be shared by all web interfaces (*provisioning*, *administrator interface* and *customer self care interface*), or separate ones for each them can be used.

- Generate the certificates. The *customer self care interface* certificate should be signed by a certification authority to avoid browser warnings.
- Upload the certificates to the system
- Set the path to the new certificates in */etc/ngcp-config/config.yml*:
  - *ossbss→apache→sslcertfile* and *ossbss→apache→sslcertkeyfile* for the *provisioning interface*.
  - *www_admin→apache→sslcertfile* and *www_admin→apache→sslcertkeyfile* for the *admin interface*.
  - *www_csc→apache→sslcertfile* and *www_csc→apache→sslcertkeyfile* for the *customer self care interface*.
- Apply the configuration changes with *ngcpcfg apply*.

8.4 Backup and recovery

8.4.1 Backup

The CE can be integrated with most of the existing backup solutions. While it does not provide any backup system by default, any Debian compatible system can be installed. It's not the scope of this chapter to go through backup system configuration. We'll focus on which information needs to be saved.

The minimum set of information to be backed up is:

- The database information.

This is the most important data in the system. All subscriber information, billing, CDRs, user preferences etc. are stored in the MySQL server. A periodical dump of all the databases should be performed.

- System configuration options

/etc/ngcp-config/config.yml, */etc/ngcp-config/constants.yml*, */etc/mysql/debian.cnf* and */etc/mysql/sipwise.cnf* files, where your specific system configurations are stored, should be included in the backup as well.

- Optional: Exported CDRs

The directory */home/jail/home/cdrexport* contains the exported CDRs the system has generated so far. It depends on your local call data retention policy whether or not to remove these files after exporting them to an external system.

- Optional: Custom files

Any custom configurations, like modified templates or additionally implemented services which are not provided by the CE.

8.4.2 Recovery

In the worst case scenario, when the system needs to be recovered from a total loss, you only need 4 steps to get back online:

- Install the CE as explained in chapter 2.
- Restore *config.yml*, *constants.yml*, *debian.cnf* and *sipwise.cnf* from the backup, overwriting your local files.
- Restore the database dump.
- Execute *ngcpcfg apply*.