FreeSWITCH PBX Example

About

This document presents a short tutorial that allows you to start using a FreeSWITCH™ server as a basic PBX. It is part of the minimal FreeSWITCH configuration which is available at https://github.com/voxserv/freeswitch_conf_minimal

Stanislav Sinyagin has graciously permitted us to publish this useful example here and we thank him for his work. The original document is on github at https://github.com/voxserv/freeswitch_conf_minimal/blob/master/docs/tutorial_01_simple_pbx.md

Introduction

> This document does not try to cover all aspects of FreeSWITCH configuration and maintenance, and it is highly recommended to read the book first: https://www.packtpub.com/networking-and-servers/freeswitch-12

After installing the minimal configuration, your FreeSWITCH server is able to process SIP requests, but its dialplan is empty, so the calls would not go anywhere. This short tutorial lists the steps to get started with a simple PBX configuration.

Further in this document, we refer to the standard FreeSWITCH configuration as "vanilla". The vanilla configuration introduces a dialplan that demonstrates lots of FreeSWITCH features, but it takes too much time to clean it up for your future production configuration. Also the vanilla configuration aliases all domains to the server's IPv4 address, making the domain name part in user registrations indistinguishable. The minimal configuration enables the "multi-tenant" scenario, where the domain name part of SIP address makes a difference. Even if you're not planning multiple domains on our FreeSWITCH server, a multi-tenant configuration still has its benefits. One of the benefits is that you can mix SIP users that connect via IPv4 and IPv6 in the same domain and let them communicate to each other.

DNS Configuration

First of all, you need to choose a domain name for your SIP service. Or even better, two different domain names: 1) for internal users to use for SIP client registration; 2) for external SIP peers to send unauthenticated calls to your server.

In this example, we use `int.example.net` as the domain name for internal SIP client registrations, and `pub.example.net` as the domain name for external peers to connect to our server.

Thus, the SIP clients would use accounts like `701@int.example.net` for registering on our server, and external peers would use a SIP URL like `sip:attendant@pub.example.net` to place unauthenticated calls to our server.

If you don't plan to receive calls via SIP URL from external peers, the DNS entry for unauthenticated calls is not necessary (although some ITSPs use this to accept calls on DID numbers).

Most modern SIP clients look up first a NAPTR DNS record in order to find out the SIP service that is serving the domain. Some DNS hosting providers (`godaddy.com`, for example) do not allow adding NAPTR records via their DNS editing GUI. A simple solution would be to point an NS record for a sub–domain to some alternative DNS hosting, such as `dns.he.net` or `gandi.net` .

It is also not too dramatic if there is no NAPTR record for your domain. Most clients fall back to the SRV record if they don't find a NAPTR record for the SIP domain.

Also, if a Windows server is used as a DNS resolver in your LAN, the NAPTR record queries may produce unpredictable results. On one occasion, I had to remove the NAPTR record from a domain, because a Gigaset C610IP phone failed to resolve the service while Windows server was used as the default resolver.

A NAPTR record should point to one or several SRV DNS records. The standard allows you to put a TCP SRV record with a higher priorty, but not all SIP clients would understand that. FreeSWITCH supports TCP transport for SIP, listening on the same ports as the UDP transport.

By default, FreeSWITCH uses port 5060 for authenticated SIP requests, and port 5080 for non-authenticated ones. Thus, the SRV records for `int.example.net` should point to UDP or TCP port 5060, and use port 5080 for `pub.example.net` records.

The following example creates the records in a BIND name server:

BIND config

```
;;; inside the zone file for example.net
pbx01               IN A 198.51.100.10
pbx01               IN AAAA 2001:DB8::0A
_sip._udp.int       IN SRV 10 0 5060 pbx01
int                 IN NAPTR 110 100 S SIP+D2U "" _sip._udp.int
_sip._udp.pub       IN SRV 10 0 5080 pbx01
pub                 IN NAPTR 110 100 S SIP+D2U "" _sip._udp.pub
```

Dialplan Contexts

The FreeSWITCH dialplan consists of contexts — independent sets of matching rules and actions for the calls. Each call enters a context, and later it may be transferred to another context, or bridged with some remote party, or a dialplan application can be executed on it according to the matching rules and actions.

The vanilla configuration defines two dialplan contexts: "public" is where all unauthenticated calls are landing, and "default" where calls to and from registered users are processed.

The minimal configuration defines only the "public" context, leaving you the freedom to define other contexts as needed.

The structure and contents of a dialplan are described in detail in this FreeSWITCH wiki and in the FreeSWITCH book. It is important to understand the two-pass processing workflow, the `continue` attribute in extensions and `break` attribute in conditions. Also you need to understand the meaning of `inlin` attribute in the action statements.

Public Dialplan Context

The file `dialplan/public/10_gateway_inbound.xml` in the minimal configuration defines a simple dispatcher for inbound calls from SIP gateways. It expects the SIP gateway to define two variables: `target_context` and `domain`, and if both are defined, the inbound call is transferred into the specified context, with the `${domain_name}` variable set to the domain name. This allows you, for example, to use multiple SIP trunks in a multi-tenant configuration, so that each trunk is used for a different tenant and its own context. The following example of a SIP gateway demonstrates the feature:

SIP gateway config

```
<!-- File: sip_profiles/external/sipcall.ch.xml -->
  <gateway name="sipcall_41449999990">
    <param name="username" value="41449999990"/>
    <param name="proxy" value="business1.voipgateway.org"/>
    <param name="password" value="xxxxxxxxxx"/>
    <param name="expire-seconds" value="600"/>
    <param name="register" value="true"/>
    <param name="register-transport" value="udp"/>
    <param name="retry-seconds" value="30"/>
    <param name="caller-id-in-from" value="true"/>
    <param name="ping" value="36"/>
    <variables>
      <variable name="domain"
                value="int.example.net" direction="inbound"/>
      <variable name="target_context"
                value="int.example.net" direction="inbound"/>
    </variables>
  </gateway>
```

Another common approach is to set up matching patterns in the public context, each matching a particular phone number or a range of numbers, and making a transfer to a specific extension:

PBX public dialplan

```
<!-- File: dialplan/public/20_inbound_did.xml -->
  <extension name="0449999990">  <!-- Hotline -->
    <condition field="destination_number" expression="^0449999990$">
      <action application="transfer" data="7000 XML int.example.net"/>
    </condition>
  </extension>
  <extension name="0449999991"> <!-- Automatic Attendant -->
    <condition field="destination_number" expression="^0449999991$">
      <action application="transfer" data="7800 XML int.example.net"/>
    </condition>
  </extension>
```

Internal Dialplan Context

In this example for int.example.net, registered users can dial 7xx to reach other registered users, 500 for an audio conference (unmoderated, anyone can join), 520 to check voicemail, and anything that starts with `0` or `1` or + goes to the PSTN. Calls to the PSTN are processed in a separate context — this is done to simplify the logic and to let you manage PSTN calls from different internal contexts in a single place.

This example dialplan allows specifying the fallback path for every extension, by defining the variable `fallback_route` in the user directory. The value of this variable should be `voicemail` for falling back to the voicemail system, or `transfer DEST CONTEXT` to transfer the call to a corresponding destination in specified context. This allows for the flexibility of defining the fallback routes for each user. Keep in mind that if you are using `mod_xml_curl` for user directory, each execution of `user_data` function will trigger the HTTP request for this user. Thus, defining and checking too many variables for a user may cause performance degradation.

Note that in the fallback transfer action, we copy the original caller ID from the variable `caller_id_number`. If the outbound connection allows us to set the caller ID, it will be preserved from the original inbound call.

PBX internal dialplan

```
<!-- File: dialplan/int.example.net.xml -->
<include>
  <context name="int.example.net">

  <!-- Call locally registered user, and fall back if the user is not registered or not answering. -->
  <extension name="Local_Extension">
    <condition field="destination_number" expression="^(7\d\d)$">
      <action application="set" data="dialed_extension=$1"/>
      <action application="set" data="dialed_user=$1@${domain_name}"/>
      <action application="set" data="ringback=${de-ring}"/>
      <action application="set" data="transfer_ringback=$${hold_music}"/>
      <action application="set" data="call_timeout=60"/>
      <action application="set" data="hangup_after_bridge=true"/>
      <action application="set" data="continue_on_fail=true"/>
      <action application="bridge" data="user/${dialed_user}"/>
      <action application="execute_extension" data="local_ext_failure"/>
      <action application="hangup" data="NO_ANSWER"/>
    </condition>
  </extension>

  <!-- Extract fallback_route from the user directory and perform corresponding actions -->
  <extension name="Local_Extension_Failure">
    <condition field="destination_number" expression="^local_ext_failure$" break="on-false">
      <action application="set" inline="true" data="fallback_route=${user_data(${dialed_user} var
fallback_route)}"/>
    </condition>

    <condition field="${fallback_route}" expression="^voicemail$" break="on-true">
      <action application="answer"/>
      <action application="sleep" data="1000"/>
      <action application="voicemail" data="default ${domain_name} ${dialed_extension}"/>
    </condition>

    <!-- transfer DEST CONTEXT -->
    <condition field="${fallback_route}" expression="^transfer\s+(\S+)\s+(\S+)$" break="on-true">
      <action application="set" data="outbound_caller_id_number=${caller_id_number}"/>
      <action application="transfer" data="$1 XML $2"/>
    </condition>
  </extension>

  <extension name="conference">
    <condition field="destination_number" expression="^500$">
      <action application="answer"/>
      <action application="sleep" data="500"/>
      <action application="conference" data="example_net"/>
    </condition>
  </extension>

  <extension name="check_voicemail">
    <condition field="destination_number" expression="^520$">
      <action application="answer"/>
      <action application="sleep" data="500"/>
      <action application="voicemail" data="check default ${domain_name} ${ani}"/>
    </condition>
  </extension>

  <!-- send the call to PSTN -->
  <extension name="pstnout">
    <condition field="destination_number" expression="^[01+]">
      <action application="transfer" data="${destination_number} XML pstnout"/>
    </condition>
  </extension>

  </context>
</include>
```

pstnout Dialplan Context

This is an example of dialplan context that places the outbound call to external telephony providers. If you are building a multi-tenant PBX, you may want to have separate dialplan contexts for each tenant, so that their calls are billed to different accounts.

This example demonstrates the rules for number normalization: the caller is supposed to be in Switzerland and is dialing a Swiss local number with one leading zero, or an international number with two leading zeros. These rules are needed to build a full E.164 number before placing the call to Voxbeam. The rules apply to called and calling numbers accordingly.

Note that we use the channel variable `outbound_caller_id_number` to determine the caller ID. This is not a standard FS variable, and it needs to be set somewhere previously in the dialplan. In our example, it is set in the user directory and in the fallback transfer condition.

pstnout dialplan

```
<!-- File: dialplan/pstnout.xml -->
<include>
  <context name="pstnout">

  <extension name="pstn_normalize" continue="true">
    <condition field="destination_number" expression="^00([1-9]\d+)$" break="never">
      <action inline="true" application="set" data="e164_dest=$1"/>
    </condition>
    <condition field="destination_number" expression="^0([1-9]\d+)$" break="never">
      <action inline="true" application="set" data="e164_dest=41$1"/>
    </condition>
    <condition field="${outbound_caller_id_number}" expression="^00([1-9]\d+)$" break="never">
      <action inline="true" application="set" data="e164_cid=$1"/>
    </condition>
    <condition field="${outbound_caller_id_number}" expression="^0([1-9]\d+)$" break="never">
      <action inline="true" application="set" data="e164_cid=41$1"/>
    </condition>
  </extension>

  <!-- here we select Voxbeam as outbound ITSP for specific country codes -->
  <extension name="pstn_select_itsp" continue="true">
    <condition field="${e164_dest}" expression="^(7|38|81)" break="on-true">
      <action inline="true" application="set" data="outbound_itsp=voxbeam"/>
    </condition>
  </extension>

  <extension name="pstn_voxbeam">
    <condition field="${outbound_itsp}" expression="^voxbeam$" break="on-false">
      <action application="set" data="effective_caller_id_number=${e164_cid}"/>
      <action application="bridge" data="sofia/gateway/voxbeam_outbound/0011103${e164_dest}"/>
    </condition>
  </extension>

  <!-- send everything else to sipcall.ch -->
  <extension name="pstn_sipcall">
    <condition field="destination_number" expression="^(0\d+)$">
      <action application="set" data="effective_caller_id_number=${outbound_caller_id_number}"/>
      <action application="bridge" data="sofia/gateway/sipcall_41449999990/$1"/>
    </condition>
  </extension>

  </context>
</include>
```

Carrier Outbound Gateway

Voxbeam uses authentication by SIP IP address, so we need to define a gateway which does not register on the provider, and sends the caller ID in the `From:` field.

Voxbeam gateway example

```
<!-- File: sip_profiles/external/voxbeam.xml -->
<include>
  <gateway name="voxbeam_outbound">
    <param name="realm" value="sbc.voxbeam.com" />
    <param name="register" value="false" />
    <param name="caller-id-in-from" value="true"/>
  </gateway>
</include>
```

User Directory

The user directory is quite standard as described here in the FreeSWITCH wiki. An important detail is that the domain name should be specified explicitly, and it should match the DNS records as described above. This is especially important in a multi-tenant environment.

You have a choice of specifying the password in clear text or as MD5 hashes for better security.

In this example, user 701 has the fallback route to a mobile number, 702 falls back into voicemail, and 703 does not have any fallback at all.

PBX user directory example

```
<!-- File: directory/int.example.net.xml -->
<include>
  <domain name="int.example.net">
    <params>
      <param name="dial-string" value="{^^:sip_invite_domain=${dialed_domain}:presence_id=${dialed_user}
@${dialed_domain}}${sofia_contact(*/${dialed_user}@${dialed_domain})}"/>
    </params>

    <variables>
      <!-- this is the context where outbound calls will start -->
      <variable name="user_context" value="fsdemo.voxserv.net"/>
    </variables>

    <groups>
      <group name="default">
        <users>
          <user id="701">
            <params>
              <param name="password" value="djrhbsdd"/>
            </params>
            <variables>
              <variable name="fallback_route" value="transfer 0794070224 pstnout"/>
            </variables>
          </user>
          <user id="702">
            <params>
              <param name="password" value="smrjbnxcvf"/>
              <param name="vm-password" value="4321"/>
            </params>
            <variables>
              <variable name="fallback_route" value="voicemail"/>
            </variables>
          </user>
          <user id="703">
            <params>
              <param name="password" value="snrfubdsf"/>
            </params>
          </user>
        </users>
      </group>
    </groups>

  </domain>
</include>
```

Firewall Settings

The following example deploys `iptables` rules that limit the rate of SIP requests to your server, thus preventing attackers from overloading your FreeSWITCH instance. Depending on your scalability requirements, the rate of allowed SIP messages can be increased. You may also have requirements for a more strict firewall policy, and add rules that reject completely certain kinds of traffic.

The following example is suitable for Debian and can be simply copy-pasted to the command line.

PBX firewall example

```
apt-get install -y iptables-persistent

#####  IPv4 rules #####
iptables -N dos-filter-sip-external
```

```
iptables -A INPUT -p udp -m udp --dport 5060 \
-j dos-filter-sip-external

iptables -A INPUT -p tcp -m tcp --dport 5060 \
-j dos-filter-sip-external

iptables -A INPUT -p udp -m udp --dport 5080 \
-j dos-filter-sip-external

iptables -A dos-filter-sip-external \
-m hashlimit --hashlimit 5/sec \
--hashlimit-burst 30 --hashlimit-mode srcip \
--hashlimit-name SIPMSG --hashlimit-htable-size 24593 \
--hashlimit-htable-expire 90000 -j RETURN

iptables -A dos-filter-sip-external -j \
REJECT --reject-with icmp-admin-prohibited

iptables -N dos-filter-ssh

iptables -I INPUT -p tcp -m tcp --dport 22 \
--tcp-flags FIN,SYN,RST,ACK SYN -j dos-filter-ssh

iptables -A dos-filter-ssh -m hashlimit --hashlimit 3/min \
--hashlimit-burst 10 --hashlimit-mode srcip,dstip \
--hashlimit-name ssh_hash --hashlimit-htable-expire 60000 \
-j ACCEPT

iptables -A dos-filter-ssh -j DROP

iptables-save > /etc/iptables/rules.v4

#####  IPv6 rules #####

ip6tables -N dos-filter-sip-external

ip6tables -A INPUT -p udp -m udp --dport 5060 \
-j dos-filter-sip-external

ip6tables -A INPUT -p tcp -m tcp --dport 5060 \
-j dos-filter-sip-external

ip6tables -A INPUT -p udp -m udp --dport 5080 \
-j dos-filter-sip-external

ip6tables -A dos-filter-sip-external \
-m hashlimit --hashlimit 5/sec \
--hashlimit-burst 30 --hashlimit-mode srcip \
--hashlimit-name SIPMSG --hashlimit-htable-size 24593 \
--hashlimit-htable-expire 90000 -j RETURN

ip6tables -A dos-filter-sip-external -j \
REJECT --reject-with icmp6-adm-prohibited

ip6tables -N dos-filter-ssh

ip6tables -I INPUT -p tcp -m tcp --dport 22 \
--tcp-flags FIN,SYN,RST,ACK SYN -j dos-filter-ssh

ip6tables -A dos-filter-ssh -m hashlimit --hashlimit 3/min \
--hashlimit-burst 10 --hashlimit-mode srcip,dstip \
--hashlimit-name ssh_hash --hashlimit-htable-expire 60000 \
-j ACCEPT

ip6tables -A dos-filter-ssh -j DROP

ip6tables-save > /etc/iptables/rules.v6
```

Conclusion

After following this tutorial, you would get a basic PBX which allows the SIP users to register and place and receive calls. It also demonstrates various features, such as a fallback scenario when the SIP user is not available.

There are many other topics which might be interesting to implement, such as:

1. Receiving and sending faxes;
2. Deploying WebRTC to let the users use their browsers as VoIP clients;
3. Setting up a static IVR;
4. Programming the server to look up some databases before connecting a call;
5. Queues, call centers, virtual assistants, etc.

See Also

Some useful scripts and configurations are available on Github and in my blog:

- https://github.com/voxserv/
- https://github.com/xlab1
- https://txlab.wordpress.com/tag/freeswitch/
- https://github.com/voxserv/freeswitch_conf_minimal/blob/master/docs/tutorial_01_simple_pbx.md — Original source of this article by Stanislav Sinyagin