# Application Note

TELECOMM SKILLS AT YOUR HANDS

# Advanced Call Control using HTTP Services and RestAPI

# with Ingate SIParator® SBC

Introduction ..... 3
About Ingate SIParator® SBC product family. ..... 3
SIParator® Call Control ..... 4
RestAPI Architecture ..... 6
The RestAPI loopback model with SIParator® ..... 7
Deployment scenarios ..... 7
Use case description. ..... 7
Proof of Concept Topology ..... 8
Configuring SIParator® SBC ..... 10
Pre-requisites ..... 10
Configuring Interface ..... 11
Other Network related configurations ..... 13
Configuring SIP in SIParator® ..... 14
Setup SIP Ports ..... 14
Setup SIP Filtering ..... 14
Setup SIP Monitoring ..... 15
Setup Call Control for RestAPI ..... 16
Configure HTTP Services to provide RestAPI Server behavior ..... 17
Configure Dial Plan ..... 22
How easy is to maintain this Configuration? ..... 27
Semi-manual update ..... 27
Automated Updates option ..... 29
Disclaimers ..... 30
Help and Support ..... 30

# Introduction

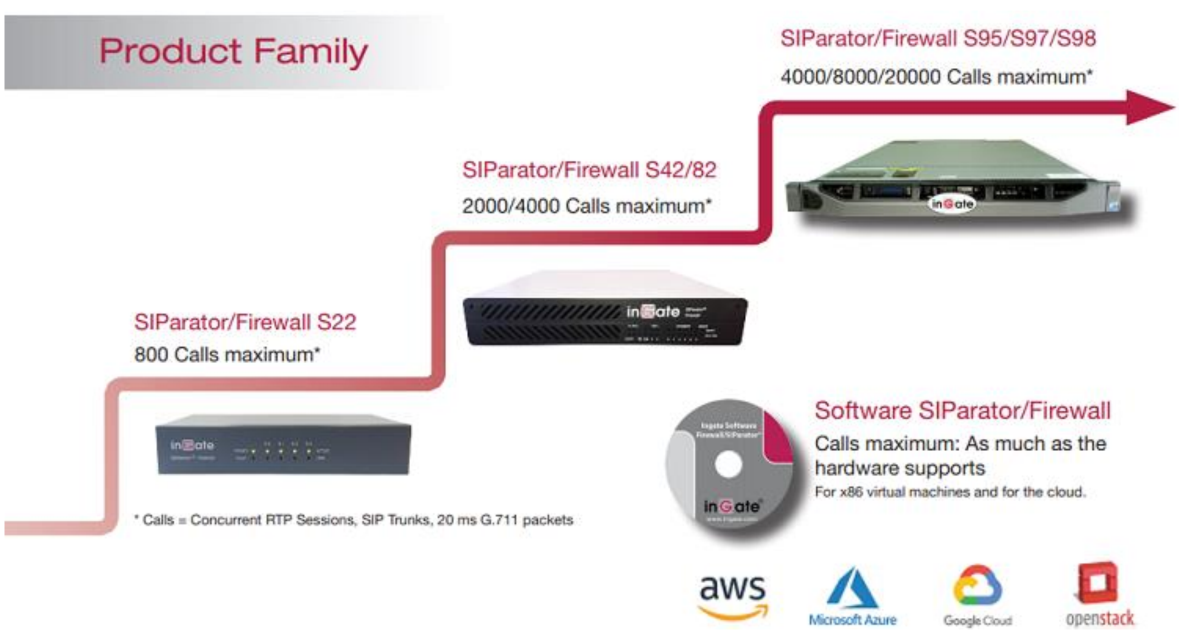
## *About Ingate SIParator® SBC product family.*

A Session Border Controller is a device that connects to an existing network firewall to seamlessly enable SIP communications (Session Initiation Protocol). While traditional firewalls block SIP traffic – including mission-critical applications like Voice over IP (VoIP) – the Ingate SIParator® SBC resolves this problem, working in tandem with your current security solutions.

The Ingate SIParator® is a powerful, flexible and cost-effective Enterprise Session Border Controller (E-SBC) for SIP connectivity, security and interoperability, such as connecting PBXs and Unified Communications (UC) solutions to SIP Trunking service providers.

The Ingate Firewall®, which is always included in the product, makes the Ingate SIParator an all-in-one appliance for data security as well as session border control.

Ingate's SIParators®/Firewalls® are available in a range of models:



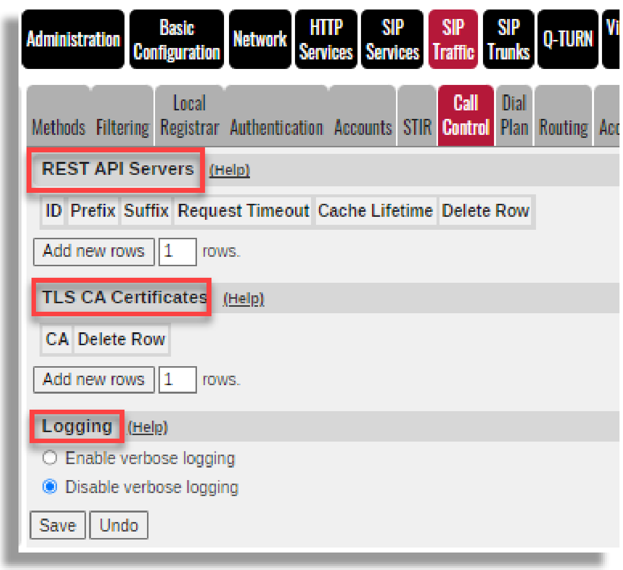

The SIParator simplifies SIP trunking and makes it easy to connect remote UC end points, aggregate SIP trunks and distribute sessions between sites and service delivery points. It's utilized for Real-Time communications security, SIP interoperability and extensive connectivity. The SIParator® is compatible with all existing networks and comes with a standard SIP proxy and a SIP registrar. It has support for NAT and PAT as well as for TLS and SRTP to encrypt both SIP signaling and media, eliminating the security issue most associated with using enterprise VoIP.

The flexible system of add-on licenses allows any enterprise to enhance the SIParator®/Firewall® solution to meet their needs at any given moment.

With more than 10,000 installations worldwide, the Ingate SIParator® comes in a wide range of capacities, and has been used by retail companies, financial institutions, industrial firms, government agencies, call centers and small-to-large enterprises.

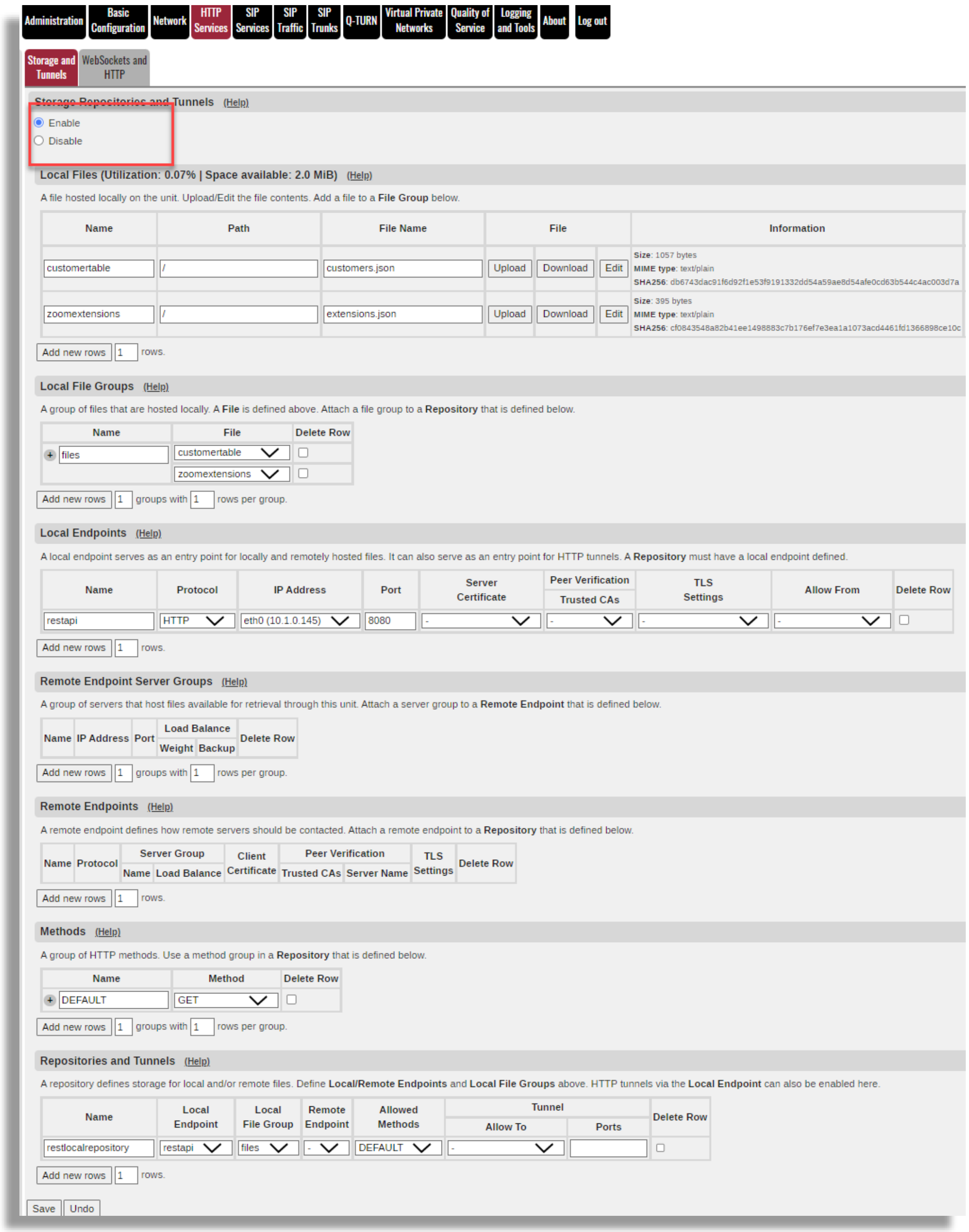
## *SIParator® Call Control*

Since version 6.2 SIParator added a new functionality to enable RestAPI client and be able to execute HTTP Requests to a Web Service during call setup to take routing decisions in real time.




Here you can define REST API servers for call control features in the Dial Plan. We will expand later in this document about this functionality.

Lately with SIParator® Release 6.4 Ingate introduced HTTP Services features, which allows to use the SIParator® as an HTTP Proxy frontend.



We will also expand on this new feature later in this document.

Now, what if we start thinking on combining Call Control RestAPI and HTTP Service? In other words, what could be the benefit of using the RestAPI client call control and use the SIParator® itself also as a RestAPI Server.

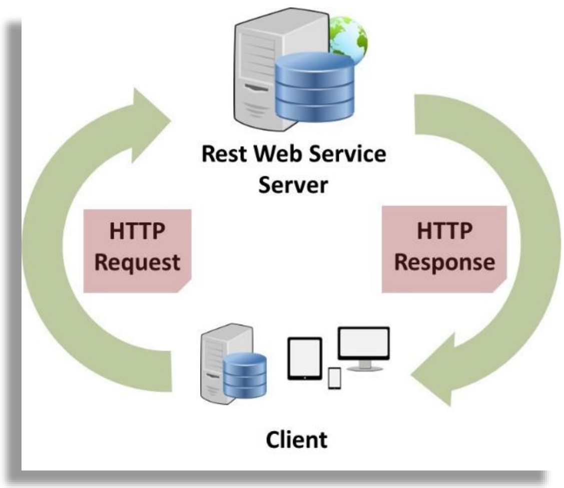
***The main purpose of this document is to illustrate new venues to build powerful logics to solve complex routing decisions with very flexible dial plans.***

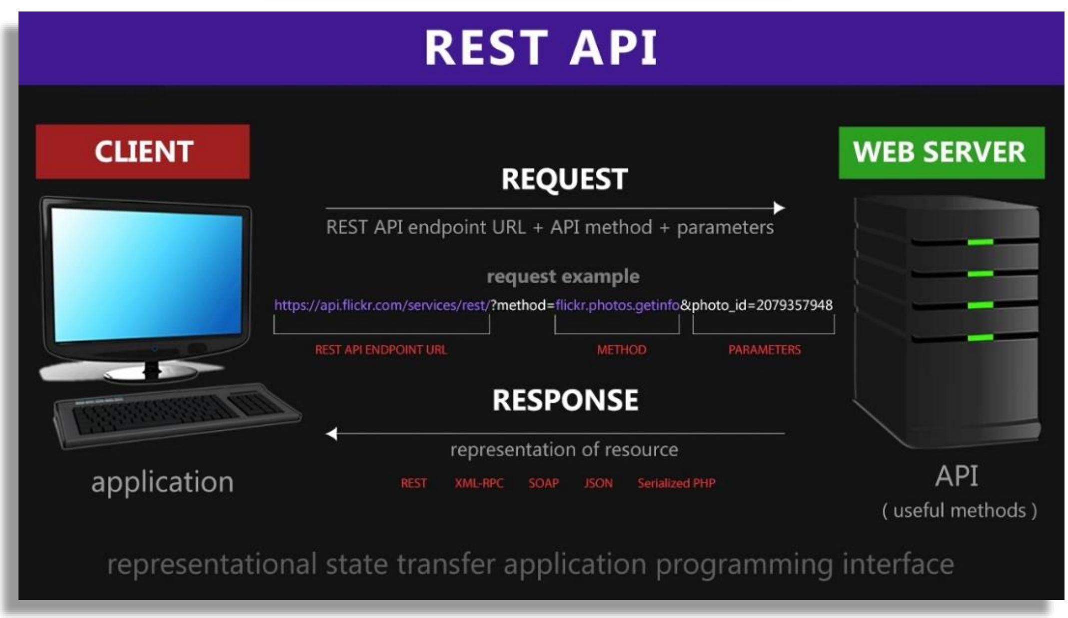
## *RestAPI Architecture*

Representational state transfer (REST) is a style of software architecture. As described in a dissertation by Roy Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web. RESTful is typically used to refer to web services implementing such an architecture.

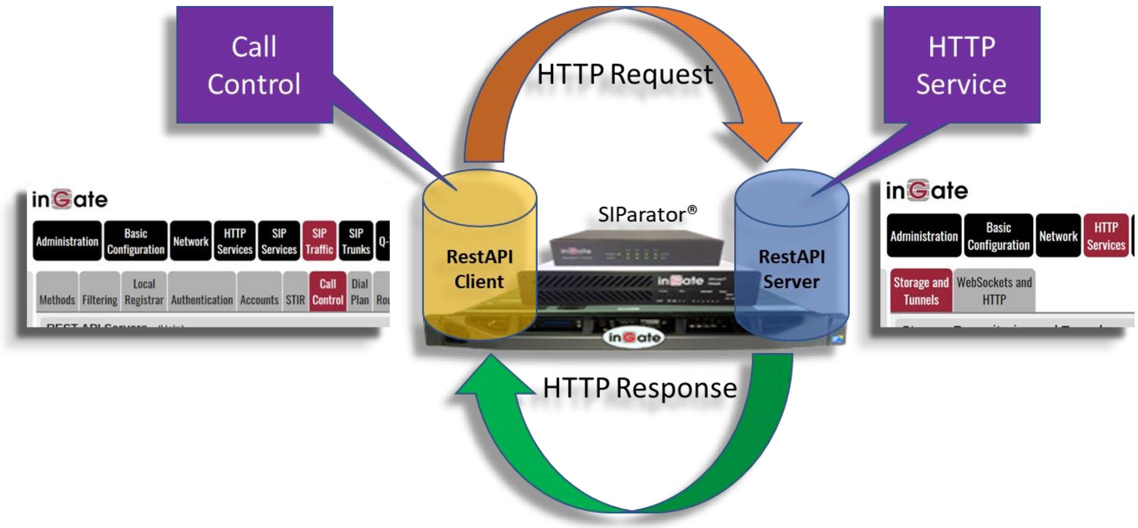
Nowadays, REST is used for integration of totally autonomous systems or application. It is vendors preferred way to open their platform to third parties or even end user's creativity.

Today eBay, Salesforce, Amazon, Cisco, and many more consider REST API as a key component of their platforms.



### *The RestAPI loopback model with SIParator®*




In our case, SIParator® will play both roles (RestAPI Client and RestAPI Server)

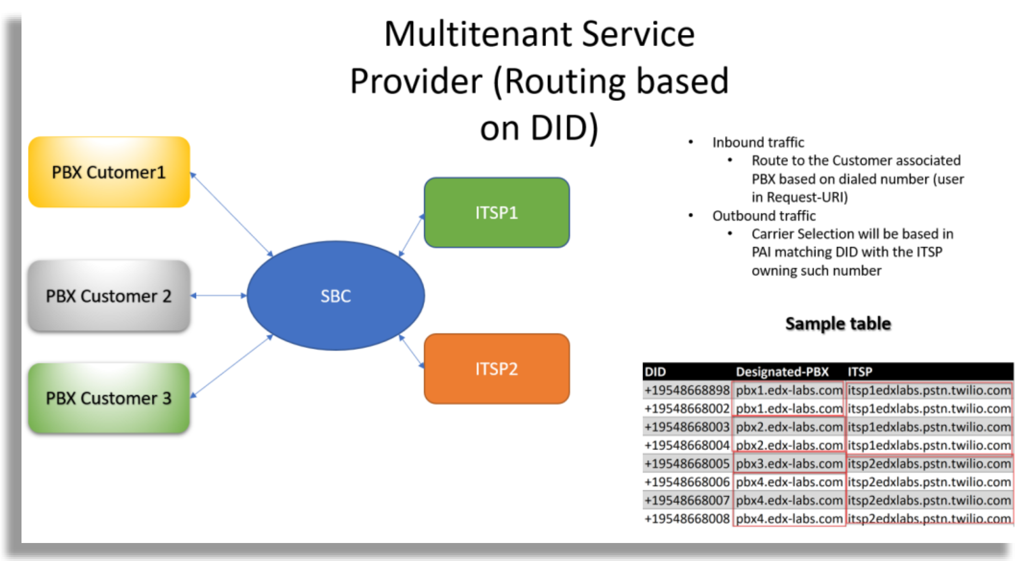
From now on we are going to call this architecture "**Loopback RestAPI**"

# Deployment scenarios

In this section we will introduce the specifics of the use case we are going to use to illustrate our "**Loopback RestAP**I"

## *Use case description.*

The case we are going to use for this illustration is a Multitenant Service Provider that is offering Hosted PBX services as well as PSTN brokerage.




| DID | Designated-PBX | ITSP |
|---|---|---|
| +19548668898 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668002 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668003 | pbx2.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668004 | pbx2.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668005 | pbx3.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668006 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668007 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668008 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |

For our example we are assuming there will be 3 customer's PBXs and 2 ITSP (SIP Trunk Providers)

Each customer has one or more DID's associated, and can also own more than one PBX.

This table is a good way to present all possible combinations of PBX, DID's and ITSPs.

| DID | Designated-PBX | ITSP |
|---|---|---|
| +19548668898 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668002 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668003 | pbx2.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668004 | pbx2.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +19548668005 | pbx3.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668006 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668007 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668008 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |

What this document will show later is how to implement a way to access this table in real-time during call setup from the dial plan.

We have to remember that there are other ways to deploy the routing for this case, but we want to illustrate how to implement with a more flexible approach. Having the routing rules based in a table such the one presented here, allows us to easily implement changes, additions or updates by just updating the table accordingly.

## *Proof of Concept Topology*

Our lab to proof concept this case is shown below:

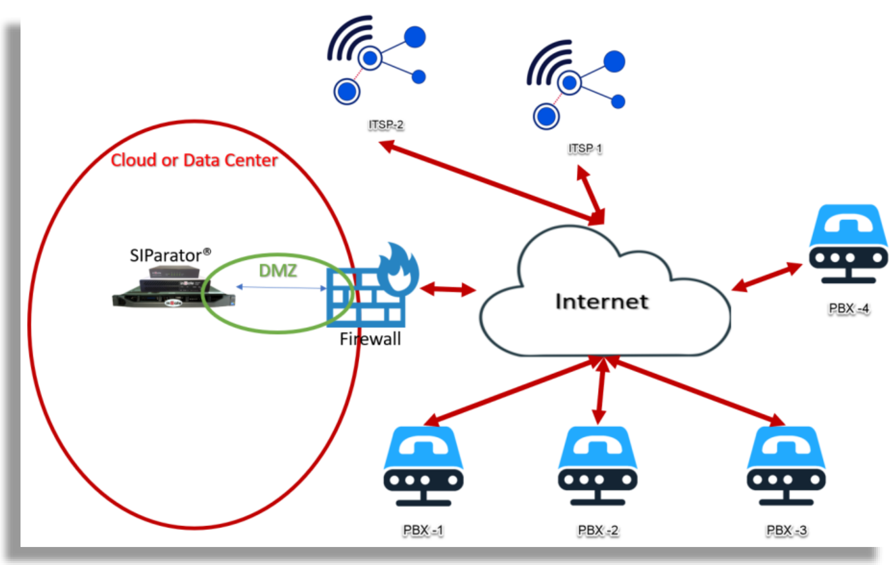



*Figure 1: Deployment Layout*

SIParator® will be deployed in a Data Center (In our case AWS Cloud) and ITSPs as well as PBXs will be located somewhere in the Public Internet

Also SIParator® will be behind AWS firewall in a public Subnet (DMZ).

Customer's PBXs are reachable via FQDN, in our example, and based in our sample table:

- pbx1.edx-labs.com
- pbx2.edx-labs.com
- pbx3.edx-labs.com
- pbx4.edx-labs.com

Service Providers (ITSPs) will be reachable also via FQDN. In our example even we are using the same provider (Twilio in our case – [www.twilio.com](http://www.twilio.com) ), we will have separated trunks for each ITSP in our table:

- itsp1edxlabs.pstn.twilio.com
- itsp2edxlabs.pstn.twilio.com

# Configuring SIParator® SBC

## *Pre-requisites*

For this use case, validation has been done running SIParator® release 6.4.1 and the minimum licensing needed must include:

- Number of sip trunk concurrent session. Also known as CCS and must be at least the maximum number of concurrent SIP sessions we want the solution to support.
- Using the “Loopback RestAPI” approach we will not use Trunk Groups as all the calls will be managed directly in the Dial Plan.
- As we are going to use HTTP Services, at least 1 ACL license is needed to enable the HTTP Feature. Additional ACL might be needed only is SIP over WebSockets (WS or WSS) is used with Registrar Method, which is not our case.

If you have any doubts or questions about the best options for licensing, feel free to send your questions to support@educronix.com

No other licenses are needed to this specific use case. When transcoding is needed, there are no license needed as Transcoding feature is a built-in functionality purely based on software.

Make sure you are using one of the SIParator® appliances according to your expected workload, or a VM properly dimensioned if you are using Software SIParator®

Before initiating the deployment make sure you have:

- A Public IP address to be used exclusively for your SBC. It can be assigned in your firewall and properly routed to the SIParator® DMZ ip address.

### Configuring IP Network Interfaces

SBC Interfaces will be assigned IP addresses for

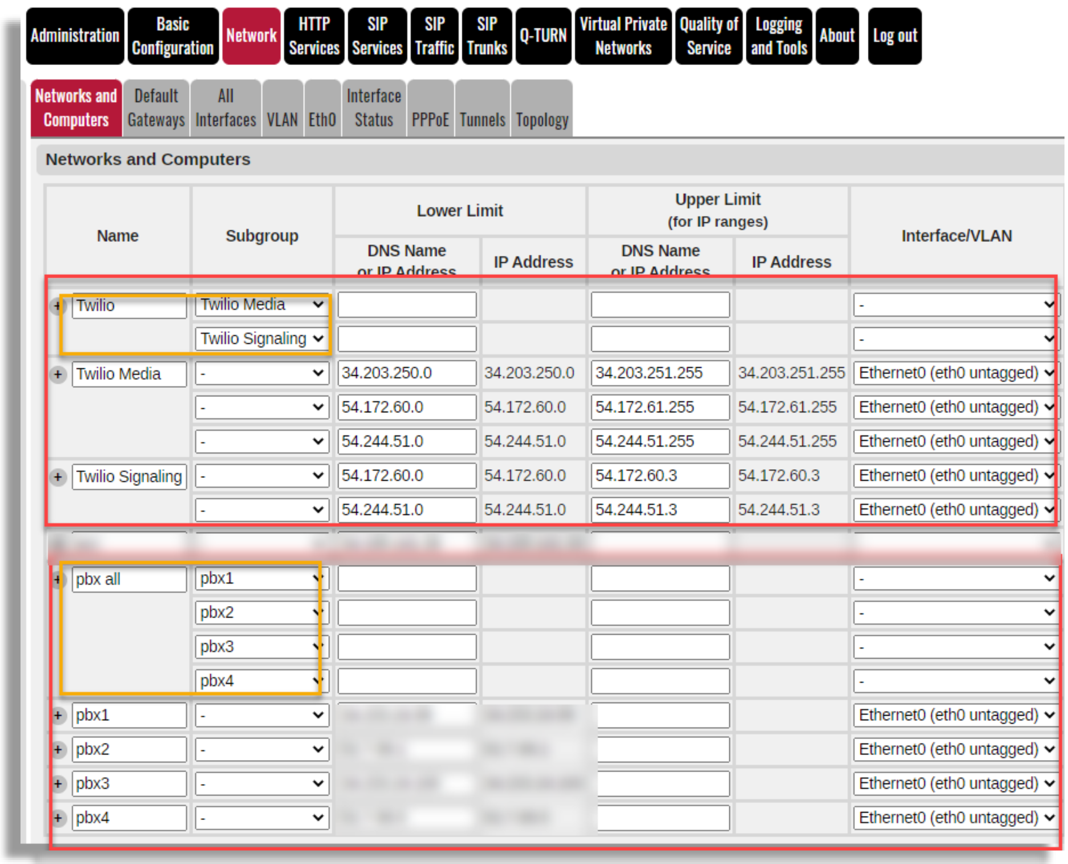
- Outside Interface. The one sitting in the DMZ and associated to the public IP address.
- For the purpose of this use case, where ITSPs and PBXs are remotely located entities, no additional interface is needed.

SBC, in our case, is connected to the WAN/Internet through a DMZ connection.

In our case all interfaces are dedicated ethernet ports.

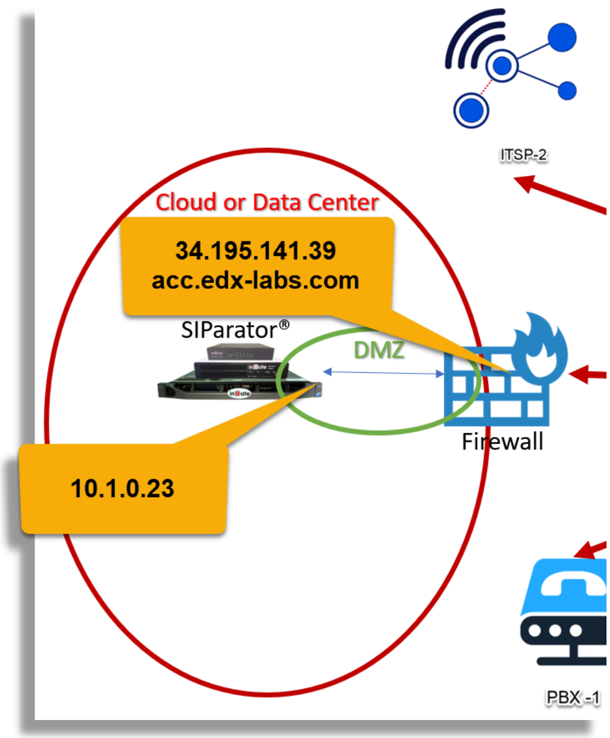
## *Configuring Interface*

First, we will assign names to known IP addresses and ranges easily used later in the configuration. By known addresses we mean SIP Proxy addresses and Media Addresses for Customer's PBXs as well as ITSPs.




- For PBXs we created an entry name for each Customer PBX IP address and then we created a name (pbx all) to aggregate all PBXs.
- For Twilio (ITSPs) we created a range for SIP Signaling and other ranges for Media, and then aggregated all of them under "Twilio" name.
- You should adapt to the ITSPs you are connecting to as well as PBXs.

Looking at our topology:




In our case,

- DMZ Network: 10.1.0.0/24
- Default Gateway: 10.1.0.1
- Eth0 IP 10.1.0.23
- Public IP: 34.195.141.39
- Public FQDN: acc.edx-labs.com

**Directly Connected Networks** (Help)

| Name | Address Type | DNS Name or IP Address | IP Address | Netmask / Bits | Network Address | Broadcast Address | Interface or Tunnel | VLAN Id | VLAN Name | Delete Row |
|---|---|---|---|---|---|---|---|---|---|---|
| eth0 | Static | 10.1.0.23 | 10.1.0.23 | 24 | 10.1.0.0 | 10.1.0.255 | Ethernet0 (eth0) | | - | |

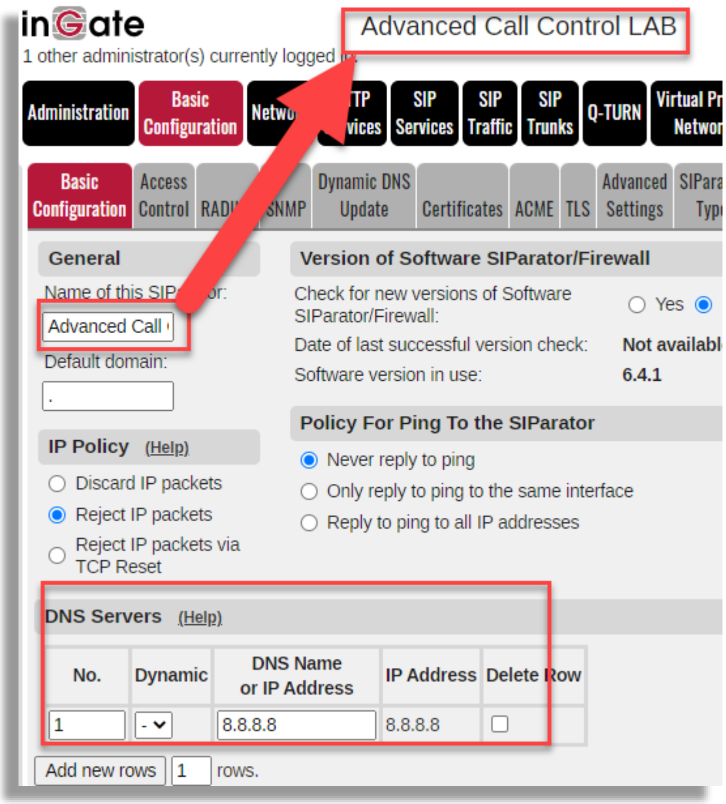
Static route for the default gateway:

**Static Routing** (Help)

| Routed Network | | | Router | | | Interface or Tunnel | Delete Rov |
|---|---|---|---|---|---|---|---|
| DNS Name or Network Address | Network Address | Netmask / Bits | Dynamic | DNS Name or IP Address | IP Address | | |
| default | default | | - | 10.1.0.1 | 10.1.0.1 | Ethernet0 (eth0) | |

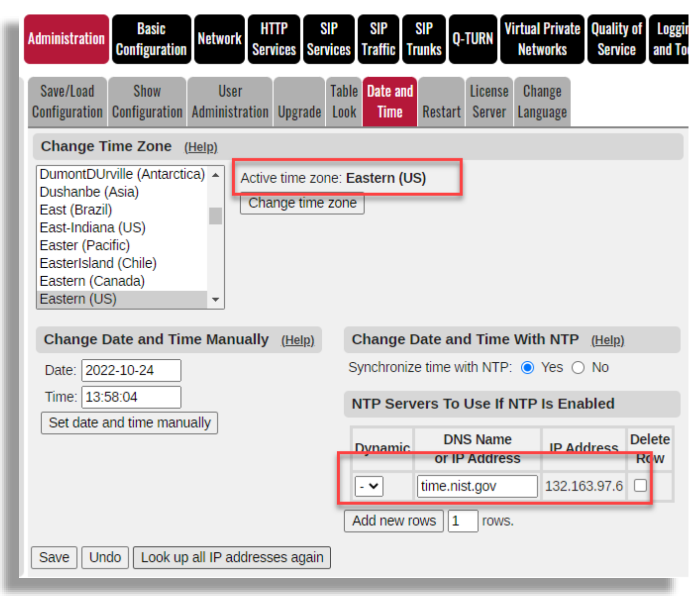
## Other Network related configurations

Let's assign the DNS server address. In our case we are going to use Google DNS 8.8.8.8




You can also assign a name to this SIParator. The name will displayed in your browser tags.

Let's also assign an NTP server and setup time for the SIParator®. We are assuming to be located in EST time zone.



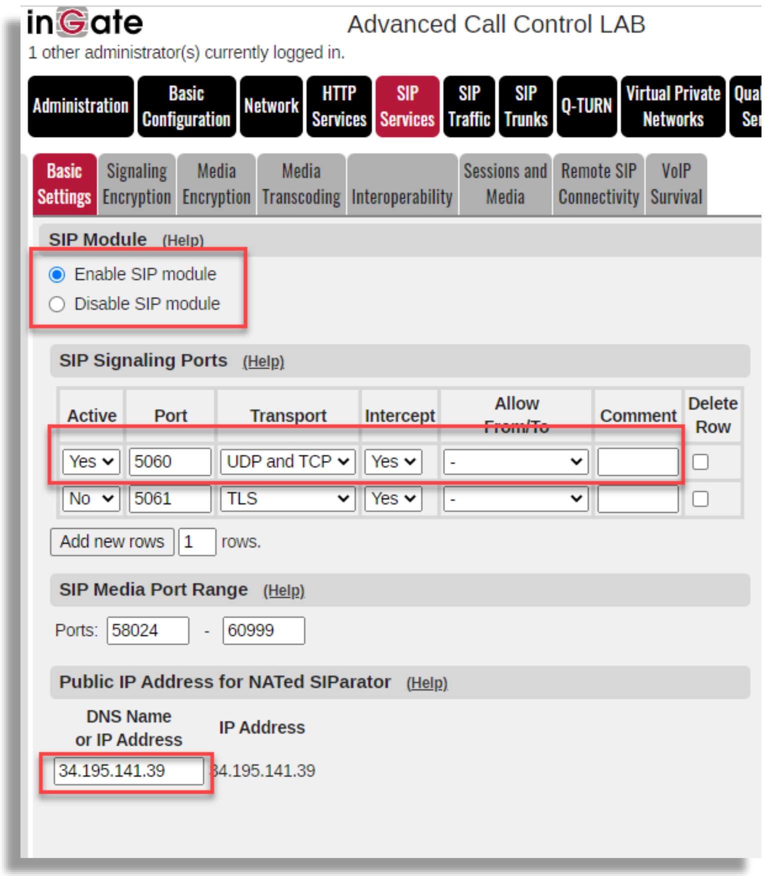
## *Configuring SIP in SIParator®*

Now we will setup all signaling related configuration for SIP.

### *Setup SIP Ports*

Now we will need to associate ports to be used for SIP (UPD/TCP and/or TLS)

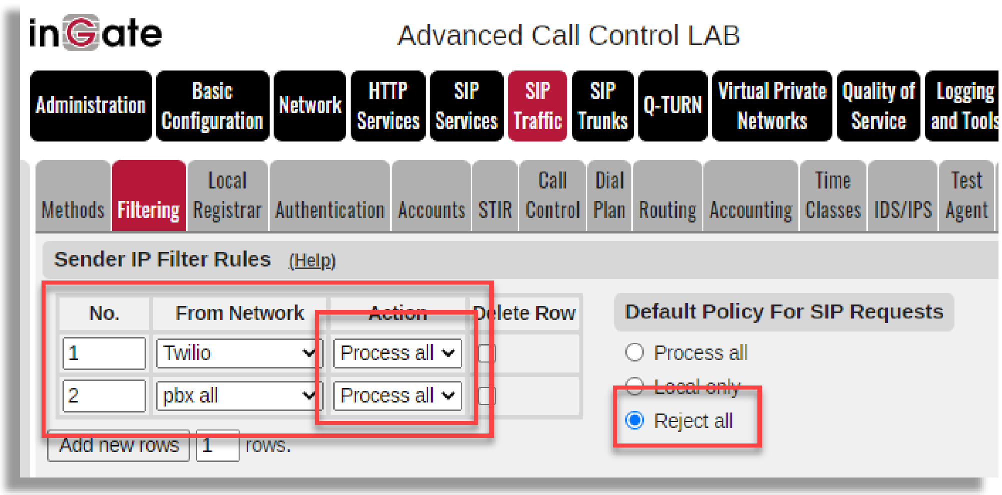
Go under SIP Services → Basic Settings




- Make sure SIP Module is enabled
- By default, SIP Signaling port 5060 for UDP and TCP is already enabled and "Allow from" enables access from any network. We can later restrict this for only sources we trust for UDP or TCP.
- Port 5061 for TLS is non active. We are not going to activate it for this use case.
- As our SIParator® is sitting in a DMZ, the public IP is NATed and we need to write down the public IP address as indicated.

### *Setup SIP Filtering*

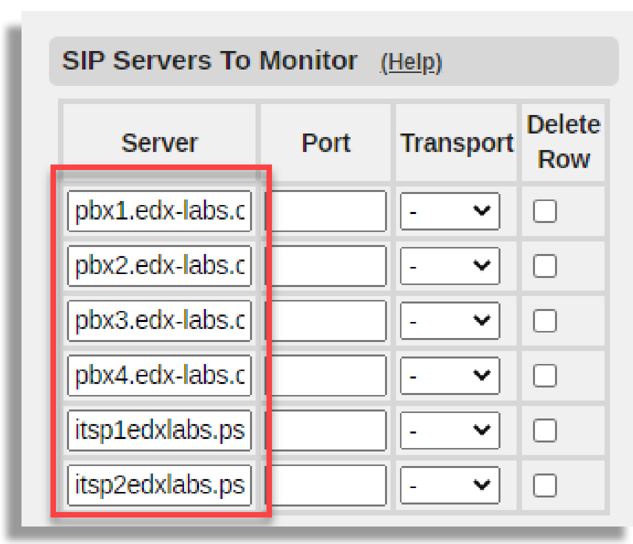
At this point we will enable SIP filtering to allow SIP traffic only from known sources




- Allow (Process all) SIP traffic from Twilio (ITSPs) as well as any of customer's PBXs.
- Reject anything else

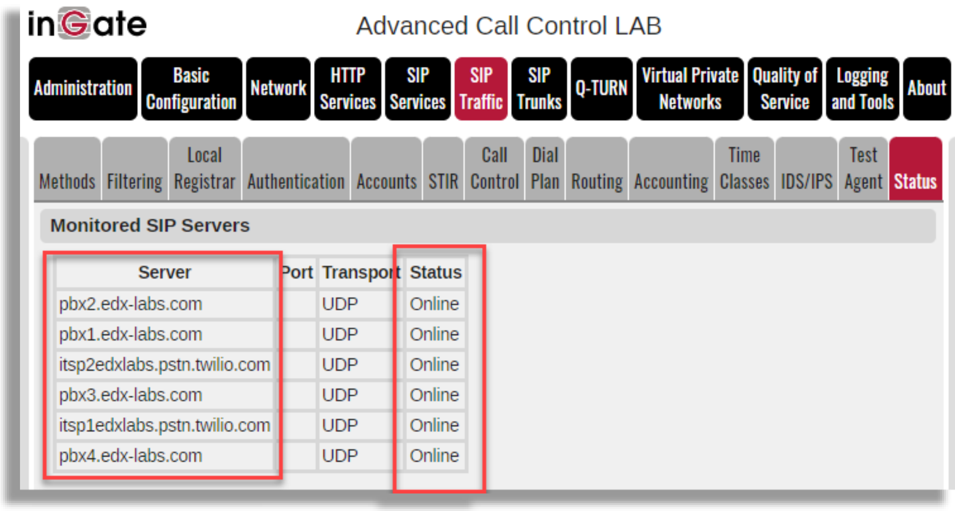
### *Setup SIP Monitoring*

At this point we also want to monitor other SIP destination/origination IP addresses. In our case we will be monitoring the ITSPs as well as the PBXs IPs. SIParator® will monitor those IP's by sending periodically SIP OPTIONS requests.




- Add all PBX IPs or FQDNs
  pbx1.edx-labs.com
  pbx2.edx-labs.com
  pbx3.edx-labs.com
  pbx4.edx-labs.com
- All all ITSPs
  itsp1edxlabs.pstn.twilio.com
  itsp2edxlabs.pstn.twilio.com

Once all destinations (PBXs and ITSPs) are setup to accept traffic from this SIParator®, you will see “online” status for each one of them.




## *Setup Call Control for RestAPI*

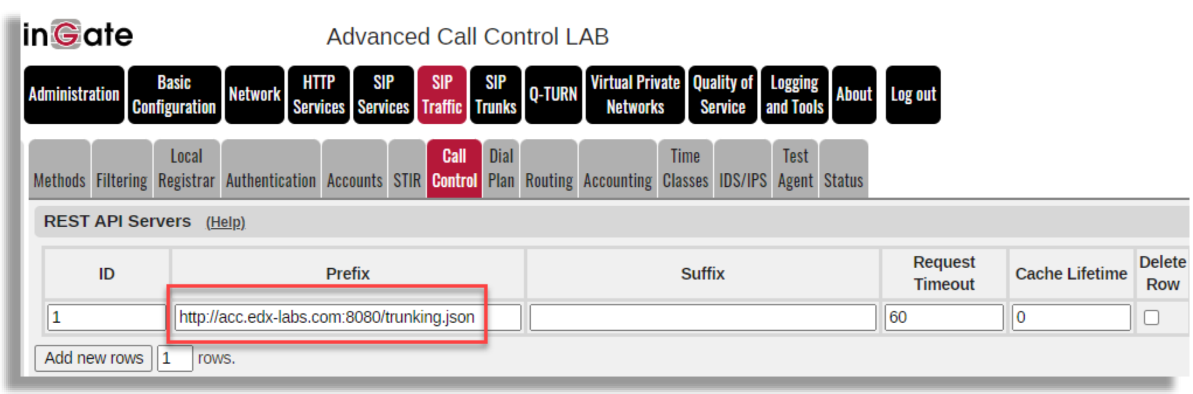
In this case we want to enable the Ingate to send Restfull API requests to a Web Service. Also, in our case the Web Service server happens to be the same SIParator® (Itself)

We have enabled an FQDN that also resolves on the public IP associated to this SIParator®

(acc.edx-labs.com)

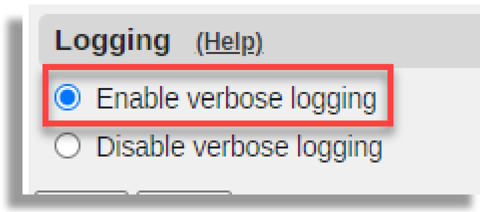
It is clear then that this FQDN points to the SIParator® itself.

In order to avoid conflicts to other services hosted by the SIParator® in port tcp 80 (i.e. ACME), we will use port 8080 instead for RestAPI Serevr functionality.



- We have selected ID=1 (to be able to refer to $curl1 function from the dial plan. It will always replace http://acc.edx-labs.com:8080/trunking.json as a prefix to any expression in the forward-to in the dial plan.
- Leave all other fields with default preloaded values.

For debugging purposes enable verbose logging.

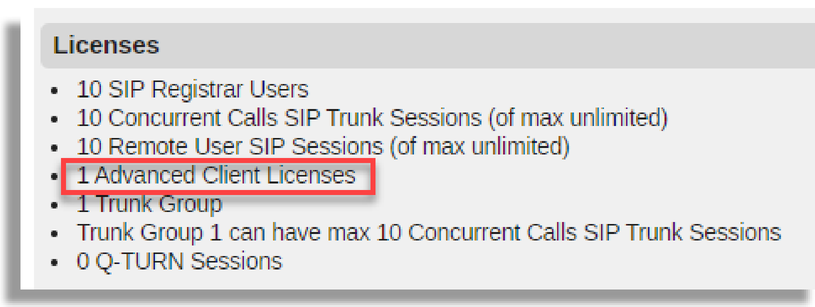



**At this point we have configured the RestAPI Client Role**

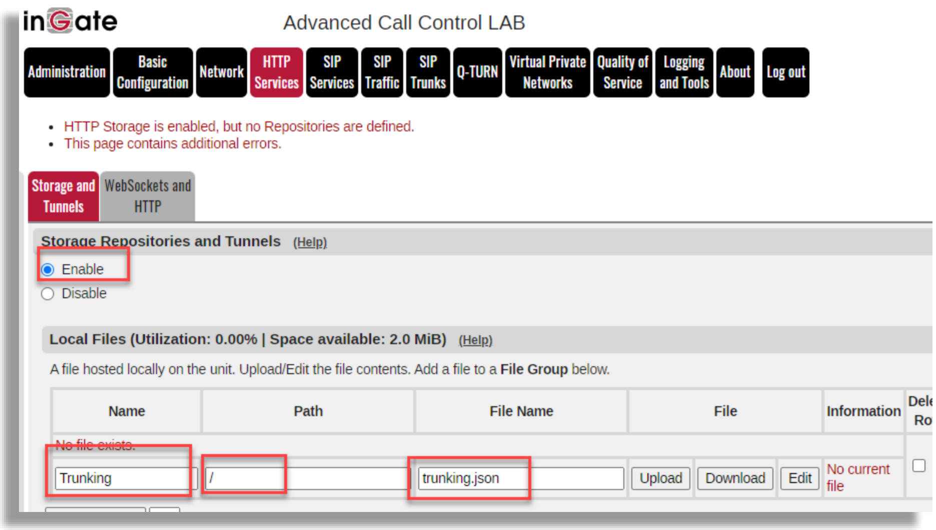
## *Configure HTTP Services to provide RestAPI Server behavior*

You should have already installed at least 1 ACL License to be able to ebale HTTP Services in the Management interface

You can check so under the About tag:




Enable HTTP Services and all a new row in the “Local Files” Section:



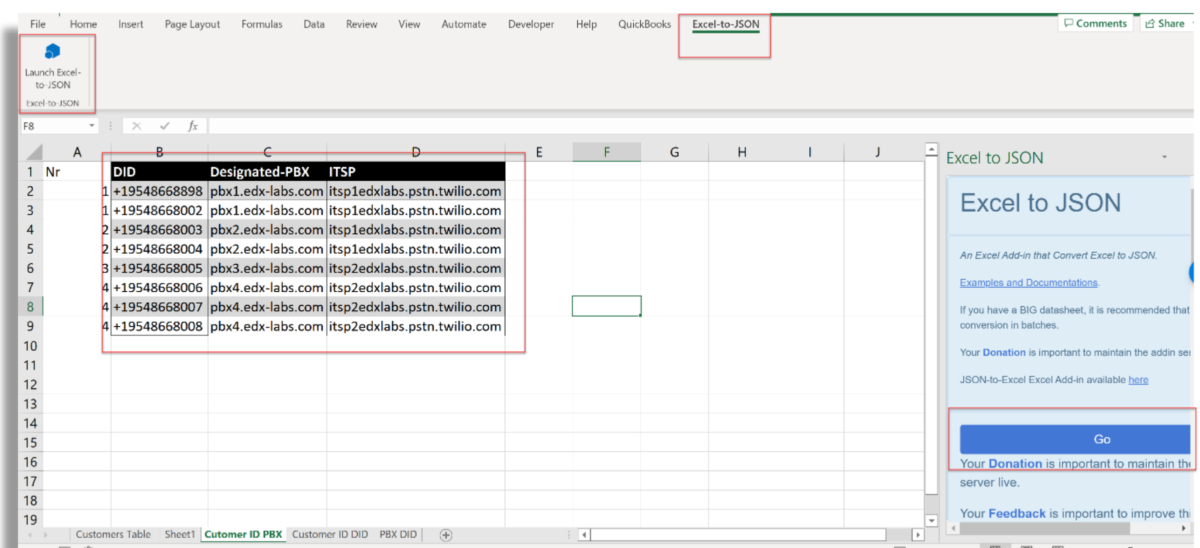
- Enable Storage Repositories and tunnels
- Add a new raw and assign a name (It could be any name with no blank spaces)
- Select the root folder (/)
- Assign a name to the file as it will be named inside the folder (trunking.json) which matches with the name used in the Call Control prefix for curl1.
- We name it trunking.json as we are going to use a json script content

To create the content and load it in the file we will take advantage of our excel file shown before.

There are many ways you can enter or create a json file, but iin our case we are using Excel and also a public add-in you can install in your Excel application.

To obtan and install this add-in you can go here: [Get Started — Excel-to-JSON 1.4.0.0 documentation (wtsolutions.cn)]

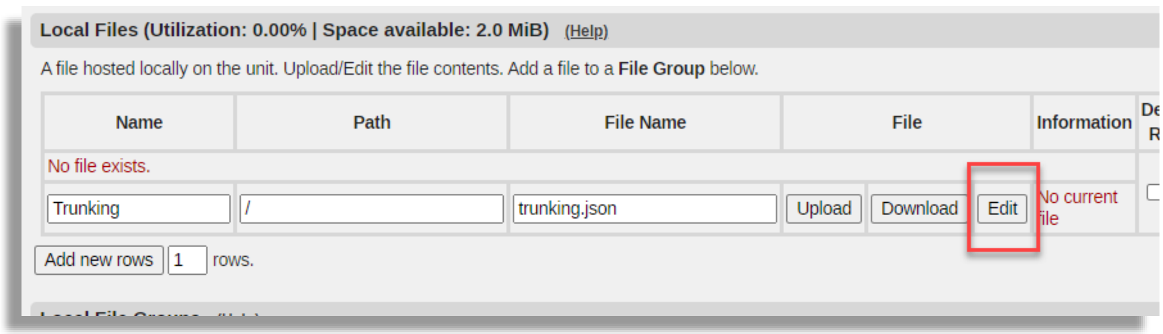
Once it is installed, you can convert your table to json script.

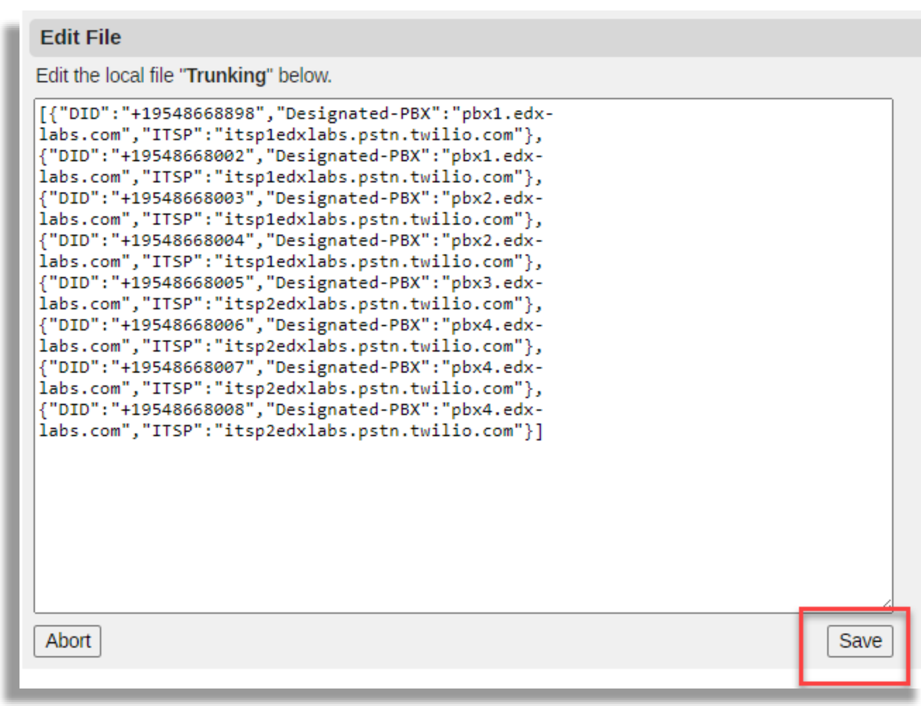


- Select Excel-to-JSON tag
- Click on Launch button
- Select the table
- Click on “Go” button

Scroll down the Excel to Json screen in the left side of the spreadsheet. You’ll see the jscon script generated and also a “ Copy to Clipboard” button.

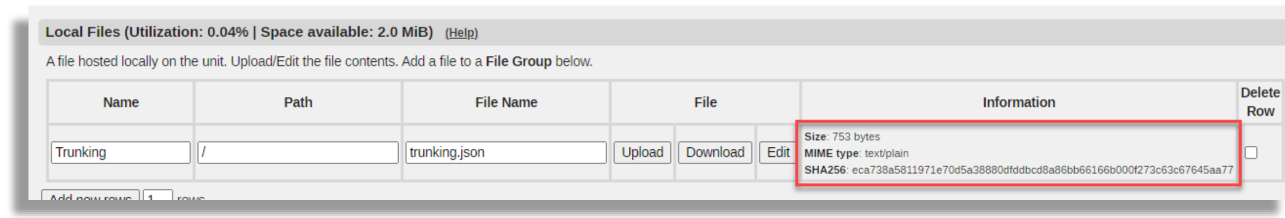
In your recenty=ly add file in the SIParator, click on the edit button, and paste there the content to created with the Excel Add-in.



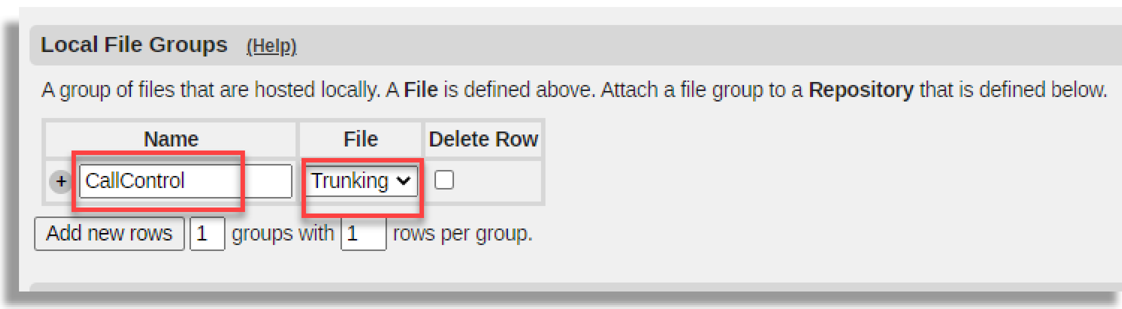

- Paste the content in the edit screen.
- Save the file

The file is now created




If you prefer you can use any other method to generate the JSON content. Here you can also use XML if it is of your preference. SIParator supports both (JSON or XML content) for call control purposes.

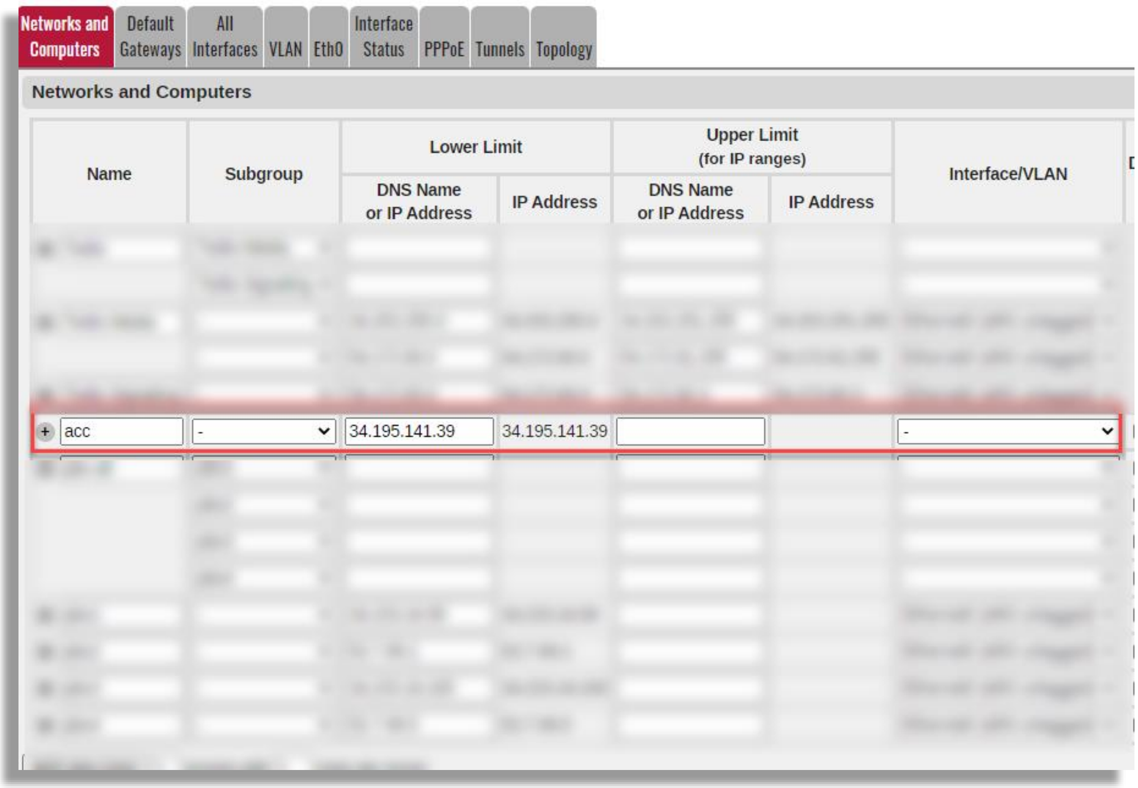
Next step is to create a local file group that contains the recently created file.



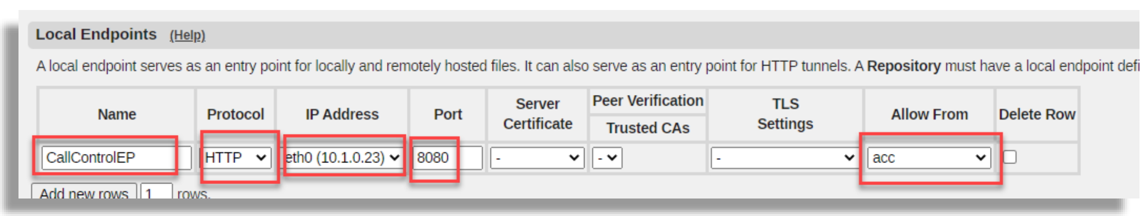

- Assign a name.
- Pull down and select he file we just created

Now, we need to create the local endpoint to enable access to this repository via port 8080 as we decided before.

Before doing that we will create a network name associated to the public IP of the SIParator® to restrict this RestAPI request only coming from itself.



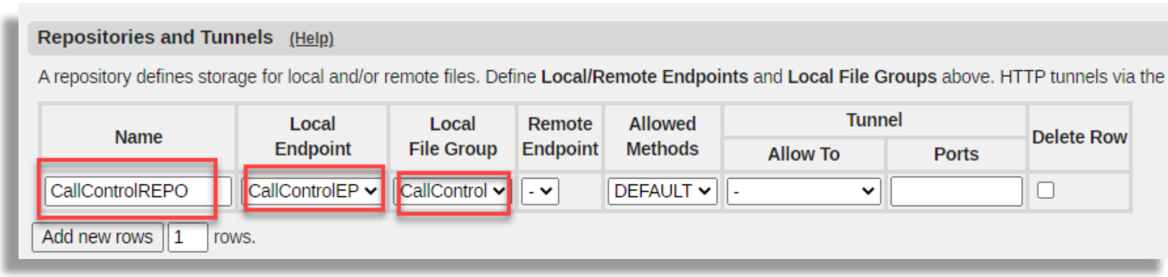

Now let's add the local endpoint




- Add a row
- Assign a name (i.e. CallControlEP
- Select the protocol (HTTP in our case)
- Select the network interface where the RestAPI requests will be served
- Restrict access only from acc network recently created

Finally let's create the repository hosting the file and how it will be accessed.

Go to Repositories and Tunnels section.




- Add a new row
- Assign a name (i.e. CallControlEP)
- Select the Local Endpoint to use
- Select the File Group to expose

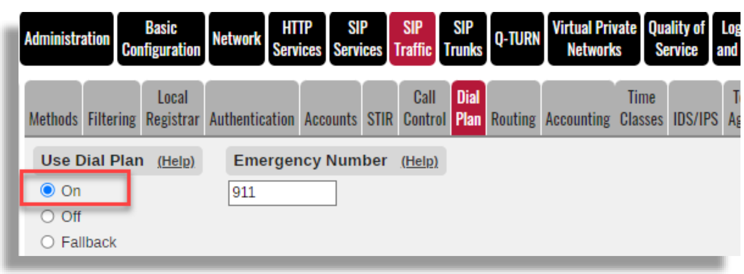
## At this point we have configured the RestAPI Server Role.

We are ready now to move to de Dial Plan configuration and setup

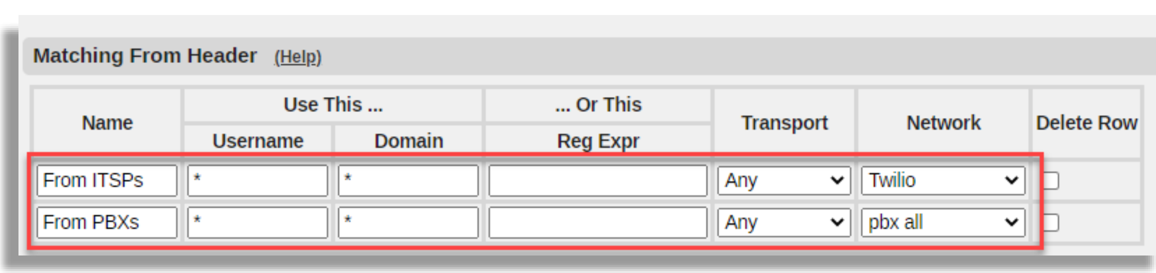
## *Configure Dial Plan*

Using Dial Plan we will be able to route inbound and outbound traffic. It includes traffic from any ITSP and properly route to the associated Customer's PBX based on number dialed from the originator (DID in the Request-URI) as well as traffic sent from any Customer's PBX to the appropriate ITSP based on the number contained in the P-Asserted-Identity in the outbound call.

First you'll need to enable Dial Plan.



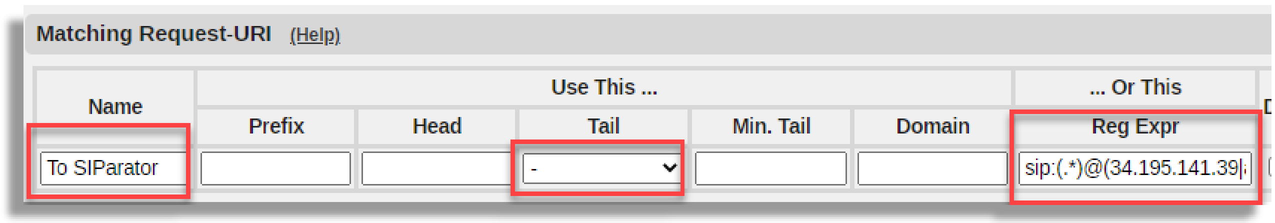

Then we need to create Matching rules for From header and Request URI. This will help on building routing rules.




- Add one row to match SIP traffic coming from the ITSP

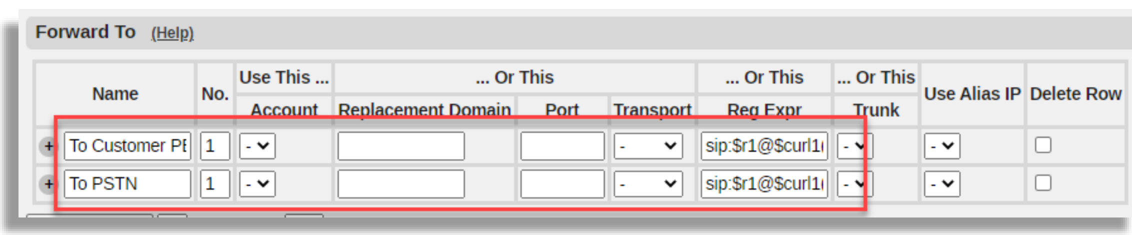
- Assign a name to each one
- We'll use wildcards ("*") for Username and Domain
- You can select specific transport protocol, but in our case we'll keep it open to "Any".
- Restrict to the Network names where the traffic could be coming from

Now, let's complete Request-URI matching




- Add one raw and assign a name
- Select "-" for Tail
- Use regular expressions and match either with the Public IP of the SIParator® or the FQDN.
  ***sip:(.*)@(34.195.141.39|acc.edx-labs.com)***

Now we will create the "Forward to" destinations, one to route traffic to Customer PBX, and the second one to the appropriate ITSP based on the phone number dialed (PSTN → PBX) or used as caller ID (PBX → PSTN)




- Add two rows
- Assign names to each one
- Use the following regular expression for PSTN → PBX

  ***sip:$r1@$curl1(_XPATH//*[DID="$r1"]/Designated-PBX/text());b2buawm;transport=udp***

- Use the following regular expression for PBX → PSTN

  ***sip:$r1@$curl1(_XPATH//*[DID="$(P-Asserted-Identity.user)"]/ITSP/text());b2buawm;transport=udp***

Lets understand in detail what we are doing with the regular expressions we just introduced.

First, we are assuming:

1) ITSP if fully compliant with E164 for any phone number used in any SIP URI (i.e. Request-URI, From, To, Contact and P-Asserted-Identity headers)
2) The PBX sends the Caller ID in the "P-Asserted-Identity" header

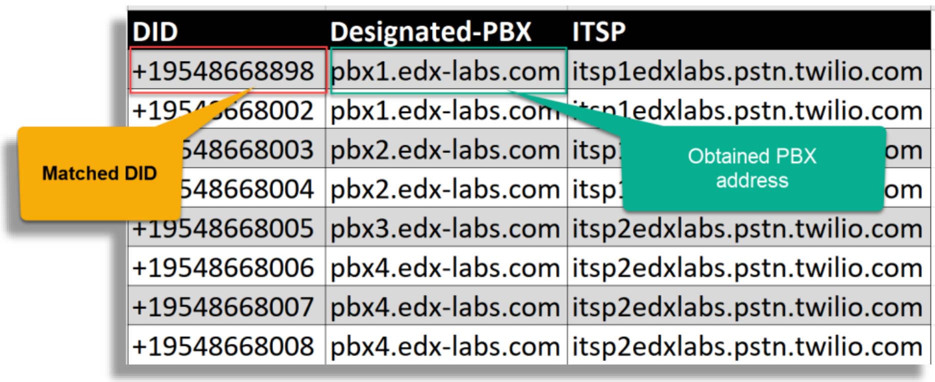
3) Reference to $Curl1 are using The Call Control definitions we created before

***sip:$r1@$curl1(_XPATH//*[DID="$r1"]/Designated-PBX/text());b2buawm;transport=udp***

- We are using XPATH navigation/parsing function supported by SIParator®, which will help navigate in our JSON file hosted by HTTP Services.
    - //* → allows us to search all levels in the file starting in the root position.
    - [DID="$r1"] → look for the row that matches DID field with the value obtained from the variable $r1. $r1 is obtained in the first parenthesis match in the Request-URI matching rule
    - /Designated-PBX/text() → describes which field will be used when matching DID field. In our case the field used is "Designated-PBX". Then "/text()" means the result will be converted to plain text.

  So, the XPATH here will look in the JSON file for the value of Designated-PBX field where the DID matches the number that was called and matched in the Request-URI.


- $curl1(…) will use the Call control to look for the web service and return the value obtained in the previous point, so the result will be = ***pbx1.edx-labs.com***



| DID | Designated-PBX | ITSP |
|---|---|---|
| +19548668898 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +195[illegible]668002 | pbx1.edx-labs.com | [illegible]1edxlabs.pstn.twilio.com |
| [illegible]548668003 | pbx2.edx-labs.com | itsp[illegible]om |
| [illegible]548668004 | pbx2.edx-labs.com | itsp[illegible]om |
| +19548668005 | pbx3.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668006 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668007 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668008 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |



After replacing curl1 result the expression will look like:

***sip:$r1@pbx1.edx-labs.com;b2buawm,transport=udp***

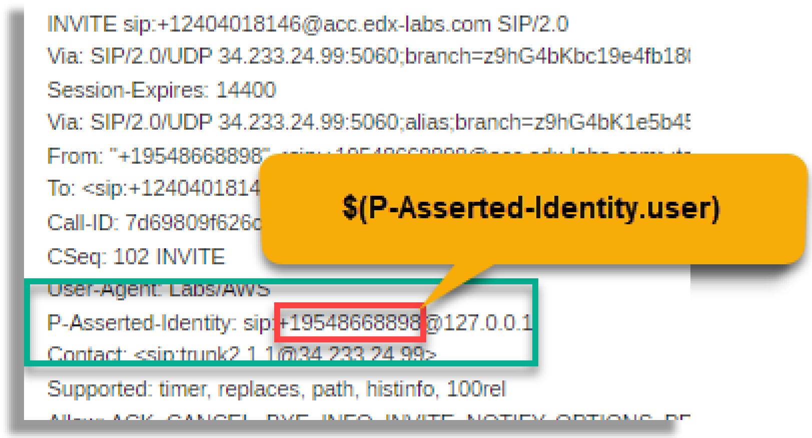
- Then $r1 will be replaced again for the dialed number +19548668898, and the final destination to route the call will be:

***sip:+19548668898@pbx1.edx-labs.com;b2bua;transport=udp***

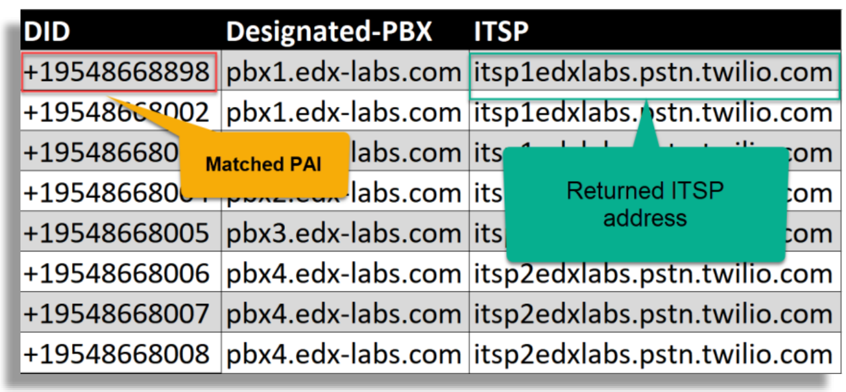
```
INVITE sip:+19548668898@pbx1.edx-labs.com;transport=udp SIP/2.0
Via: SIP/2.0/TCP 34.195.141.39:5060;branch=z9hG4bK4b5f4a8d73cb93a82af2
Session-Expires: 14400
Via: SIP/2.0/UDP 34.195.141.39:5060;branch=z9hG4bKb247ca5fd041947436f2
From: <sip:+12404018146@itsp1edxlabs.pstn.twilio.com:5060;isup-oli=62;pstn-
To: <sip:+19548668898@pbx1.edx-labs.com;transport=udp>
Call-ID: 13ca1134e292f71225b02515de8e3714@sipgt-57d6798b
CSeq: 537556 INVITE
User-Agent: SIParator/6.4.1
Contact: <sip:eiuldvKCkGABnYsiONzy6YPb1iEEligCwPJby-iir684.@34.195.141
Supported: timer, replaces, path, histinfo, 100rel
```

***sip:$r1@$curl1(_XPATH//*[DID="$(P-Asserted-Identity.user)"]/ITSP/text());b2buawm;transport=udp***

In this case, instead of looking for the $r1 (dialed number) will look for $(P-Asserted-Identity.user) value in the call coming from the PBX. It will be matched against DID field in the table.




And we will extract the value of the "ITSP" field in text format



| DID | Designated-PBX | ITSP |
|---|---|---|
| +19548668898 | pbx1.edx-labs.com | itsp1edxlabs.pstn.twilio.com |
| +195486[illegible]002 | pbx1.edx-labs.com | itsp1edxlabs.[illegible]stn.twilio.com |
| +195486680[illegible] | [illegible]labs.com | its[illegible]com |
| +19548668[illegible] | [illegible]labs.com | its[illegible]com |
| +19548668005 | pbx3.edx-labs.com | its[illegible]com |
| +19548668006 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668007 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |
| +19548668008 | pbx4.edx-labs.com | itsp2edxlabs.pstn.twilio.com |



For useful information about regular expressions and XPATH you can visit the following links:

XPath Tutorial (w3schools.com)

How_To_use_Generic_Header_Manipulation.pdf (ingate.com)

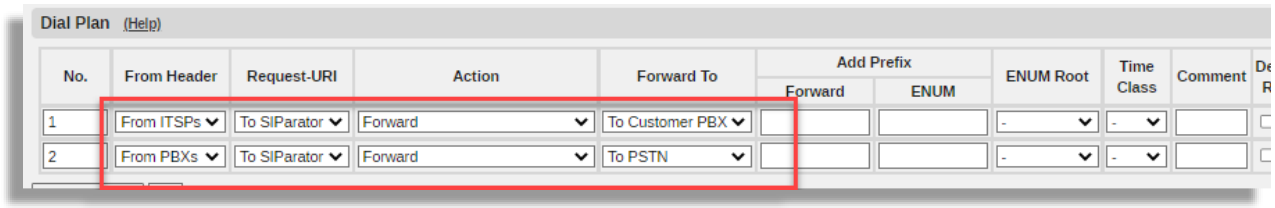
regex101: build, test, and debug regex

Ingate Reference Guide (call Control)

Ingate Reference Guide (HTTP Services)

Next step is to complete the actual dial plan.

Dial plan execution happens in the Dial Plan Table



Dial Plan (Help)

| No. | From Header | Request-URI | Action | Forward To | Add Prefix Forward | Add Prefix ENUM | ENUM Root | Time Class | Comment | De R |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | From ITSPs | To SIParator | Forward | To Customer PBX | | | - | - | | |
| 2 | From PBXs | To SIParator | Forward | To PSTN | | | - | - | | |

- Add two rows to the dial plan table
- The first will match From Header based on the matching rule named “From ITSPs” and match Request-URI named “To SIParator”. When that happens, call will be routed using “Forward to” named “To Customer PBX”
- The second will match From Header based on the matching rule named “From PBXs” and match Request-URI named “To SIParator”. When that happens, call will be routed using “Forward to” named “To PSTN”

# How easy is to maintain this Configuration?

## *Semi-manual update*

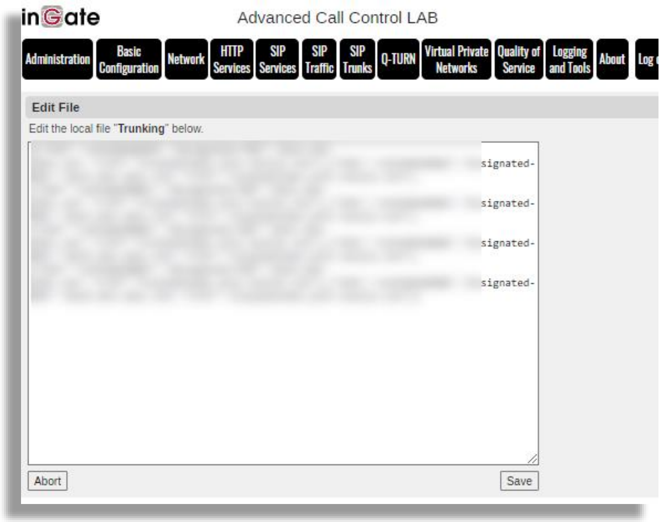
At this point, the configuration we just created is very easy to maintain. Maintenance means:

- Change attributes assigned to a given DID (i.e. change ITSP hosting it, will be associate to another PBX, etc…)
- Add new or delete DIDs.
- Add new or delete customers
- Add new or delete ITSPs
- Etc…

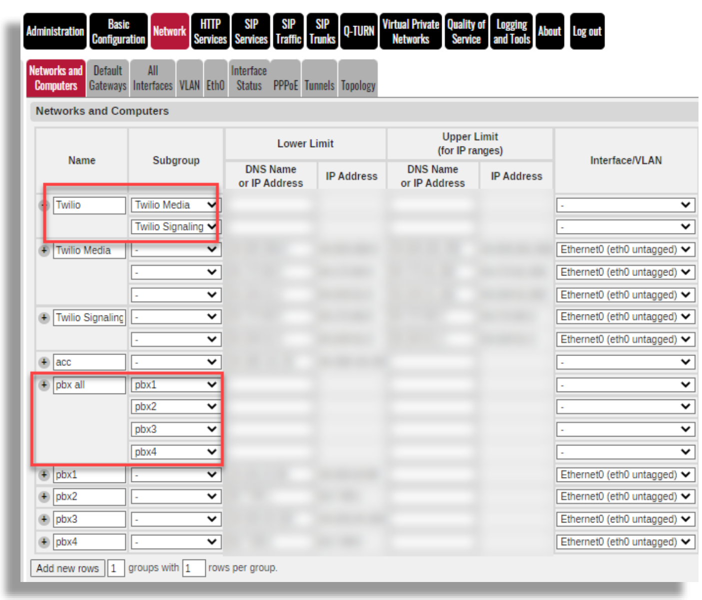
This is the first beauty of this approach (“easy to maintain”)

First, any change, add or remove rows can be done just in the table:

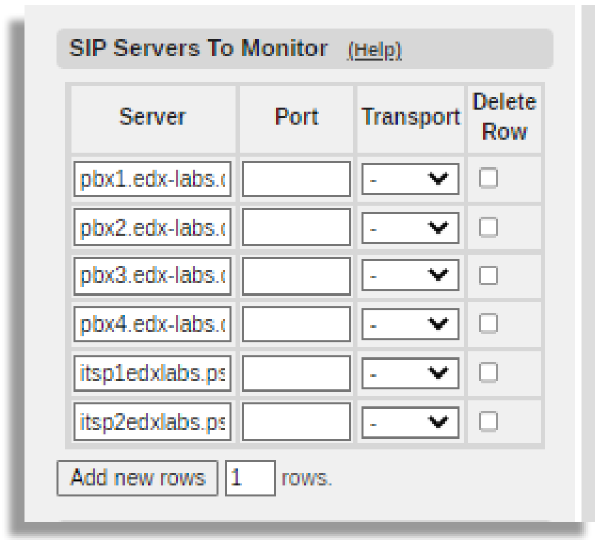
- Update your Excel file
- Execute the add-in “Excel-to-JSON”.
- Copy and paste in the HTTP Services local file.




If any of the updates done added a new ITSP address or PBX address, make the appropriate adjustments in the Network→Networks and Computers Section.




If new endpoints were added such as new PBX, or new ITSP proxy addresses, you should add them to the SIP Services → Basic Settings, under the Servers to Monitor table.




## *Automated Updates option*

In case you want to create a more automated way to update, you can do 2 things:

Instead of using a manual process to maintain an Excel table and then manual cut and paste in the HTTP Services local file section, you can implement RestAPI provisioning and use your own scripting of implement Ingate's SDK

For more details see here:

- https://account.ingate.com/manuals/6_4_1/reference_guide_6_4_1.html#_access_control (look on how to enable RestAPI clients)
- https://account.ingate.com/manuals/6_4_1/reference_guide_6_4_1.html#_python_sdk
- https://account.ingate.com/manuals/6_4_1/reference_guide_6_4_1.html#_command_line_reference

This way with some development you can automate updates to your base of customers, ITSPs etc… with a totally external application.

# Disclaimers



# Help and Support