exotel
DOCS

Exotel AgentStream

CONTENTS

1. Get Started

Exotel AgentStream is a real-time voice streaming service that connects live phone calls (PSTN or SIP) to your AI systems or bots over the internet.

- In **unidirectional streaming**, AgentStream sends audio **one way** – from the customer's phone leg to your application.
  - Use case: transcription, analytics, passive monitoring, or recording.
- In **bidirectional streaming**, AgentStream sends audio **both ways** – customer speech is streamed to your bot, and your bot's responses are streamed back into the live call.
  - Use case: interactive voicebots, agent assist, real-time translation, or conversational AI.

In simple terms:

- **Unidirectional = listen only.**
- **Bidirectional = listen + talk.**

AgentStream gives you the flexibility to pick either mode depending on whether you just need **call insights** or a **conversational AI experience**.

This guide outlines how to enable and use Exotel's AgentStream(aka Voice Streaming) capabilities via the Voicebot Applet or Stream Applet. These voice streaming applets allow real-time audio streaming between your customer calls and bot platforms over WebSocket.

Connect Voice AI API

AgentStream Applets

Connectors and Bridges

StreamKit Cloud

Voice AI Ecosystem

Connect Applet

1.1. Overview & Quickstart

Step 1: Account Setup

Before enabling voice streaming services, ensure you have an active Exotel account in the appropriate region.

Choose the Right Instance

Instance Location	URL	Use Case
Mumbai (Veeno)	Link	Required for BFSI accounts or compliance, WebSocket + SIP Trunking, or IP–PSTN intermix scenarios(Recommended for Indian Customers)
Singapore	Link	Default region for most use cases

Note: IP–PSTN intermixing is supported only on the Mumbai (Veeno) instance.

Step 2: KYC Verification Indian Accounts

AgentStream access requires completion of your KYC verification.

- Submit your company documentation as per standard onboarding.
- Refer to the complete KYC process here: How to complete KYC for Exotel
- Wait for confirmation from the Exotel compliance team before proceeding.

Step 3: Integration and Support

Once access is enabled, proceed with integration using the documentation below.

1.2. Connect Voice AI API

The Connect voice to Bot API lets you programmatically call a phone number and connect the call to a conversational AI bot in real time. Audio flows in both directions – the caller speaks to the bot, and the bot responds to the caller – over a secure WebSocket connection.

Typical use cases: AI-powered outbound campaigns, virtual customer care agents, automated surveys, and appointment reminders with live confirmation.

Before You Start

You will need:

- Your Exotel Account SID, API Key, and API Token (available from your Exotel Dashboard).
- An ExoPhone (virtual number) to use as the Caller ID.
- A WebSocket server endpoint (your bot service) that accepts audio in real time.

How It Works

1. You send an API request with the caller's phone number and your bot's WebSocket URL.
2. Exotel dials the phone number.
3. Once the call is answered, Exotel opens a connection to your WebSocket server.
4. Audio is streamed live between the caller and your bot for the duration of the call.
5. When the call ends (by either party), Exotel sends a status update to your callback URL.

API Reference

Endpoint:

```
POST /v1/Accounts/{AccountSid}/Calls/connect
```

Base URLs:

Region	Base URL
Singapore	https://api.exotel.com
Mumbai	https://api.in.exotel.com

Authentication uses HTTP Basic Auth with your API Key as the username and API Token as the password.

Parameters

Required

Parameter	Description
From	The phone number to call. Use international format (e.g., +919876543210).
CallerId	Your Exotel virtual number that appears as the caller ID.
StreamUrl	The WebSocket URL of your bot server. Must start with ws:// or wss://.
StreamType	Set this to bidirectional to enable two-way audio.

Optional

Parameter	Description
Record	Set to true to record the call. Default: false.
RecordingChannels	single (merged) or dual (separate track per side). Default: single.
TimeLimit	Maximum call duration in seconds. Up to 14,400 (4 hours).
CustomField	Any text you want attached to the call record (max 128 characters). Useful for tracking order IDs, customer IDs, etc.
StatusCallback	A URL on your server that Exotel will call with updates when the call status changes.
StatusCallbackEvents	Which events to receive: answered, terminal (call ended), or ringing. You can specify more than one. This is available only for leg1
StreamName	An optional label for the stream, up to 32 characters.

Example Request

bash

```
curl -X POST 'https://<api_key>:<api_token>@api.exotel.com/v1/Accounts/<AccountSid>/Calls/connect' \  -F 'StreamType=bidirectional' \  -F 'StreamUrl=wss://your-bot.example.com/media' \  -F 'From=+91XXXXXXXXXX' \  -F 'CallerId=0XXXXXXXXXX' \  -F 'Record=true' \  -F 'StatusCallback=https://your-server.com/callback' \  -F 'StatusCallbackEvents[]=terminal'
```

Response

A successful request returns a Call object immediately. The call is still being set up at this point – use StatusCallback to track when the call is answered and when it ends.

json

```
{  "Call": {    "Sid": "a1b2c3d4e5f6...",    "Status": "in-progress",    "From": "+91XXXXXXXXXX",  "PhoneNumberSid": "0XXXXXXXXXX",    "Direction": "outbound-api",    "DateCreated": "2025-06-01 10:00:00",  "RecordingUrl": null  }}
```

Field	What it means
Sid	A unique ID for this call. Save it to look up call details later.
Status	Current state of the call.
From	The number that was dialed.
RecordingUrl	Link to the recording once the call is complete (if recording was enabled).

Call status values:

Status	Meaning
queued	Call is being prepared.
in-progress	Call is active.
completed	Call ended normally.
failed	Call could not be placed.
busy	The number was busy.
no-answer	The number did not answer.

Configuring Your Bot's WebSocket Server

Your bot must accept a WebSocket connection and handle audio in real time. Key points:

- Exotel sends audio as raw PCM or mulaw at 8 kHz by default. To request a different sample rate, append it to your StreamUrl: wss://your-bot.example.com/media?sample-rate=16000. Supported values: 8000, 16000, 24000.
- Your bot can send audio back to the caller at any time during the call.
- When your bot wants to end the call, it closes the WebSocket connection.

For full protocol details, see the AgentStream documentation.

Things to Keep in Mind

- StreamUrl must use ws:// or wss://. Plain HTTP URLs are not supported.
- The full StreamUrl (including any query string) must be under 600 characters.
- If you provide a StreamName, it must be 32 characters or fewer.
- Recording, status callbacks, and custom fields all work the same way as they do for regular outbound calls.
- This feature must be enabled on your account before use. Contact hello@exotel.com to get started.

Need Help?

- Read the AgentStream guide to understand how Exotel streams audio to your bot.
- For API errors and response codes, see the Error Code Reference.
- Contact support at hello@exotel.com.

1.3. Connect Voice AI with Flow API

The Connect Voice AI to Flow API lets you place an outbound call that is controlled by an Exotel Flow. Flows let you design complex call experiences – bidirectional stream(Voice AI), unidirectional stream(Agent Assist), play messages, collect input, route to different outcomes – and can hand the caller off to a Agent at any point in the journey.

Use this API when your call needs logic beyond a direct bot connection: for example, playing a terms disclosure before connecting to a bot, or routing a caller to either a human agent or a bot based on their menu selection.

Typical use cases: Multi-step outbound Voice AI calls with Agent or human fallback, compliance-driven call flows, blended human + AI workflows.

Before You Start

You will need:

- Your Exotel Account SID, API Key, and API Token (available from your Exotel Dashboard).
- An ExoPhone (virtual number) to use as the Caller ID.
- A Flow built in the Exotel dashboard. If your flow includes a bot, it should have a **Voicebot** or **Stream** applet configured with your bot's WebSocket URL.

How It Works

1. You send an API request with the caller's phone number and the URL of an Exotel Flow.
2. Exotel dials the phone number.
3. Once the call is answered, the flow begins executing – playing audio, collecting DTMF, routing logic, etc.
4. When the flow reaches a Voicebot or Stream applet, Exotel opens a WebSocket connection to your bot and streams audio in real time.
5. The bot can speak to the caller and, when done, hand back to the flow or end the call.
6. Call status updates are sent to your configured callback URL.

Setting Up Your Flow

To include a bot in your flow, add one of the following applets in the Exotel Flow builder:

Applet	When to use
Voicebot	Connect the caller to an AI bot. Configure the WebSocket URL inside the applet.
Stream	Stream audio to a custom WebSocket endpoint for real-time processing.
Connect	Transfer the caller to a human agent (for bot-to-human handover).
Passthru	Pass call control to an external system mid-flow.

The StreamUrl and StreamType for the bot are configured inside the applet – you do not pass them in the API request.

API Reference

Endpoint:

```
POST /v1/Accounts/{AccountSid}/Calls/connect
```

Base URLs:

Region	Base URL
Singapore	https://api.exotel.com
Mumbai	https://api.in.exotel.com

Authentication uses HTTP Basic Auth with your API Key as the username and API Token as the password.

Parameters

Required

Parameter	Description
From	The phone number to call. Use international format (e.g., +919876543210).
CallerId	Your Exotel virtual number.
Url	Flow URL: **http://my.exotel.com/{your_sid}/exoml/start_voice/{app_id}** The App/Flow URL of the Exotel Flow that will control this call. Find this in your Flow settings in the dashboard App bazaar

Optional

Parameter	Required	Type	Description
CallType	No	String	trans for transactional calls.
TimeLimit	No	Integer	Max call duration in seconds. Max: 14400 (4 hours).
TimeOut	No	Integer	Ring timeout in seconds.
StatusCallback	No	String	Webhook URL for call status updates.
StatusCallbackEvents	No	Array	terminal, answered, or both.
CustomField	No	String	Metadata passed to the applet via Passthru.

Example Request

bash

```
curl -X POST 'https://<api_key>:<api_token>@api.exotel.com/v1/Accounts/<AccountSid>/Calls/connect' \  -d
'From=+91XXXXXXXXXX' \  -d 'CallerId=0XXXXXXXXXX' \  -d
'Url=https://my.exotel.com/<AccountSid>/exoml/start_voice/<flow_id>' \  -d 'Record=true' \  -d
'StatusCallback=https://your-server.com/callback'
```

Replace <flow_id> with the numeric ID of your Flow from the dashboard.

Response

A successful request returns a Call object immediately. Use StatusCallback to track the call as it progresses through the flow.

json

```
{  "Call": {    "Sid": "a1b2c3d4e5f6...",    "Status": "in-progress",    "From": "+91XXXXXXXXXX",
"PhoneNumberSid": "0XXXXXXXXXX",    "Direction": "outbound-api",    "DateCreated": "2025-06-01 10:00:00",
"RecordingUrl": null  }}
```

Field	What it means
Sid	A unique ID for this call.
Status	Current state of the call.
RecordingUrl	Link to the recording after the call ends (if recording was enabled).

Call status values:

Status	Meaning
queued	Call is being prepared.
in-progress	Call is active.
completed	Call ended normally.
failed	Call could not be placed.
busy	The number was busy.
no-answer	The number did not answer.

Common Flow Patterns

VoiceBot Applet → **Passthru Applet** → **Programmable Connect: Working with Connect Applet (Dynamic URL)** Place a Voicebot applet early in the flow, followed by a Connect or Transfer applet. The bot handles routine queries and escalates to a live agent when needed.

Bot → Passthru Applet Use a Passthru applet after the Voicebot applet to pass call control to your own telephony system for further handling.

IVR → VoiceBot Applet handover Design your flow with an IVR Menu applet first, then connect the appropriate option to a Voicebot applet. Callers navigate the menu, and the bot handles the selected intent.

Bot with compliance disclosure Start with a Greeting applet to play a required disclosure, then move to the Voicebot applet. The caller hears the disclosure before the bot begins the conversation.

Things to Keep in Mind

- Your Flow must be created and active in the Exotel dashboard before making API calls.
- The Url parameter must point to a valid Exotel AppEngine flow URL. Custom external URLs are not supported for this parameter.
- The Voicebot and Stream applets must be configured inside the flow with the correct WebSocket URL. Do not pass StreamUrl or StreamType directly in this API call.

- Recording, status callbacks, and custom fields work the same way as they do for regular outbound calls.

Need Help?

- Learn how to build flows with the Exotel Flow Builder guide.
- Configure your bot in a flow using the Voicebot Applet guide.
- For a direct bot connection without a flow, see the Connect to Bot API.
- Contact support at hello@exotel.com.

1.4. Programmable Voice APIs with AgentStream

When you build bot-driven voice experiences, reducing perceived latency is critical. With Exotel ExoML, you can control call legs and media actions in real time. This guide explains two recommended approaches:

- **Approach 1**: Create a customer leg and immediately start a stream.
- **Approach 2**: Create a leg, start a stream, and optionally play a short greeting (say/play) in parallel while the bot connects.

Both flows are supported with existing **ExoML – Introduction**

Prerequisites

- An Exotel account and valid account_sid, key and Token
- An exophone configured for outbound calls
- A gRPC endpoint to receive leg events.
- A WebSocket bot server that supports Exotel streaming and events

Approach 1: Create Leg + Start Stream (Bot-First)

Base URL

All ExoML Aare served from the Exotel CPaaS API host.

Base URL format

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}
```

This is the simplest flow: connect the customer leg, then hand it directly to your bot.

Step 1. Create the leg

`POST /v2/accounts/{account_sid}/legs`

Content-Type: application/json

Body

```
{
"contact_uri": "074xxxxx773",
"exophone": "0204xxxxx38",
"leg_event_endpoint": "grpc://<your-example-endpoint>",
"timeout": 30
}
```

You will receive events such as leg_connecting, leg_ringing, and finally leg_answered.

Step 2. Start the stream

On leg_answered, start the stream:

`POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/start_stream`

Content-Type: application/json

Body

```
{
"direction": "bidirectional",
"url": "wss://bot.example.com/stream",
"content_type": "audio/x-mulaw;rate=8000"
}
```

At this point, audio flows between the customer and your bot.

Approach 2: Create Leg + Start Stream + Say/Play (Parallel)

In many customer-facing scenarios, even a 1–2s delay feels like silence. To improve perception, you can play a short greeting while the stream session is warming up.

Step 1. Create the leg

Same as Approach 1.

Step 2. Start actions in parallel

On leg_answered, issue two actions:

Start stream

POST

```
POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/start_stream
{
"direction": "bidirectional",
"url": "wss://bot.example.com/stream",
"content_type": "audio/x-mulaw;rate=8000"
}
```

Start greeting (choose one):

- **Say**

POST

```
POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/start_say
{
"text": "Hello",
"loop": 0
}
```

- **Play**

POST

```
POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/start_play
{
"url": "https://example.com/contact-center-tone.wav",
"loop": 0
}
```

Step 3. Stop greeting when the stream is ready

As soon as you receive the stream_started event, stop the placeholder:

`POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/stop_say`

or

`POST /v2/accounts/{account_sid}/legs/{leg_sid}/actions/stop_play`

This ensures the bot audio seamlessly takes over without overlap.

When to Use Each Approach

Scenario	Recommended Flow
Bot-first, IVR-style journeys	Approach 1
Sales / Collections (instant feel)	Approach 2
Contact center simulation	Approach 2
Tech support bot only	Approach 1

Key Notes

- **Parallelism**: Exotel supports issuing start_stream and start_say/start_play together on the same event.
- **Greeting length**: Keep placeholder audio short (200–500 ms) so the bot can take over naturally.
- **Stop timing**: Always stop say/play when you receive stream_started.

Next Steps

- Get ExoML enabled for your account

- Explore StopSay/StopPlay timing to fine-tune transitions.

1.5. Active Stream Monitoring API

The **Active Stream Monitoring API** lets you query and track all **live AgentStream sessions** running on your Exotel account in real time.

Monitor active streams:

```
curl -X GET 'https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_sid>/ActiveStreams'
```

Replace `<your_api_key>` and `<your_api_token>` with the API key and token created by you.

Replace `<your_sid>` with your “Account sid”

Replace `<subdomain>` with the region of your account

- `<subdomain>` of Singapore cluster is @api.exotel.com
- `<subdomain>` of Mumbai cluster is @api.in.exotel.com

`<your_api_key> , <your_api_token>` and `<your_sid>` are available in the API settings page of your **Exotel Dashboard**

Sample Response:

JSON

```
{
"status": "success",
"active_streams": 12,
"max_allowed_streams": 100,
"account_sid": "Exotel"
}
```

1.6. AgentStream: WSS errors and handling

Short guide for AgentStream applets – **Stream** (unidirectional) and **Voicebot** (bidirectional) – when you host a **wss://** endpoint. Exotel **opens the WebSocket to your server**; JSON events and post-call outcomes follow the main **AgentStream** docs.

Two places errors show up

Layer	What it is	Where
WebSocket close code	Standard (RFC 6455) when the socket ends	Logs from **your WebSocket server** (the side that accepts Exotel's connection), proxies, load balancers
Streaming outcome	Success/failure + optional detail text	**Passthru** and other **callbacks** – often nested as **Stream** (same idea may appear on Status Callback depending on your setup)

They are **not** the same signal. Correlate with **CallSid** and time.

Common WebSocket close codes

Code	Meaning
1000	Normal close
1001	Endpoint going away (restart, navigate, etc.)
1002–1003	Protocol / data type problem
1006	Abnormal – often **no proper close frame** (network drop, TLS, timeout, LB idle timeout, crash). Very common; check path and timeouts first.
1007–1009	Payload / policy / size
1011	Server error while handling the connection
1012–1013	Restart / try again (depends on stack)

1005 and **1015** are reserved or synthetic in many APIs – you may still see them in logs.

Other products may use **IANA-registered application codes** (e.g. 3xxx) or **private use 4000–4999** on **your** infra – not the same as text in **Stream.Error**.

If the socket drops often: validate **wss://**, cert, **firewall / IP allowlist**, proxy **idle timeouts**, and server **cold start**. For Support: **CallSid**, **UTC time**, close **code** + **reason**, server log snippet (redact secrets).

Stream.Error in callbacks

When streaming fails, **Stream.Error** may contain a **short diagnostic string**. Treat it as **opaque** – copy the **full value** for Exotel Support; you don't need to parse it.

Interpret together with **Stream.Status**, **DisconnectedBy**, **Disposition**. If **Status** and **Error** seem inconsistent, trust the **error detail** and escalate with both.

Typical focus areas

Situation	Check
Won't connect	URL, DNS, TLS, firewall, allowlist
Times out during setup	Cold start, CPU, proxy timeouts
Dies on first events after connect	Upgrade path, auth, edge limits
Mid-call	Load, backpressure, your app latency
Voicebot-only	Bidirectional behaviour per AgentStream docs

Passthru: Stream fields (when Voicebot or Stream ran before Passthru)

Present **when the event store has them** – field names:

StreamSID, StreamUrl, Status, Duration, RecordingUrl, Error, Disposition, DisconnectedBy, DetailedStatus

Leg1RingingDuration may sit **next to** Stream, not inside it. After **Connect**, you may also get **Legs**, **DialCallStatus**, **DialWhomNumber**, etc. Some parameters are **tenant-specific** – if something is missing, check your account docs or Support.

Encoding (query vs JSON) follows your **Voice / App** integration doc.

Quick checklist

- Make Passthru handlers **idempotent** (retries, dedupe on CallSid).
- Log **CallSid**, **StreamSID**, close **code**, full **Stream.Error**.
- Return **200** quickly; don't log **credentials** in URLs.

See AgentStream Applets and the **Unidirectional / Bidirectional** pages on Exotel Docs for event shapes.

2. AgentStream Applets

2.1. AgentStream Applets

AgentStream Applets are building blocks inside Exotel that let you decide *how* audio from a live call is streamed to your bot or application over wss. Each applet controls the **direction of audio flow, event handling, and bot playback behavior**

Enabling And Disabling Streams: Unidirectional and Bidirectional Streams can be enabled for a call flow using the “Stream” and “Voicebot” Applet, respectively when creating Custom Apps in the App Bazaar.

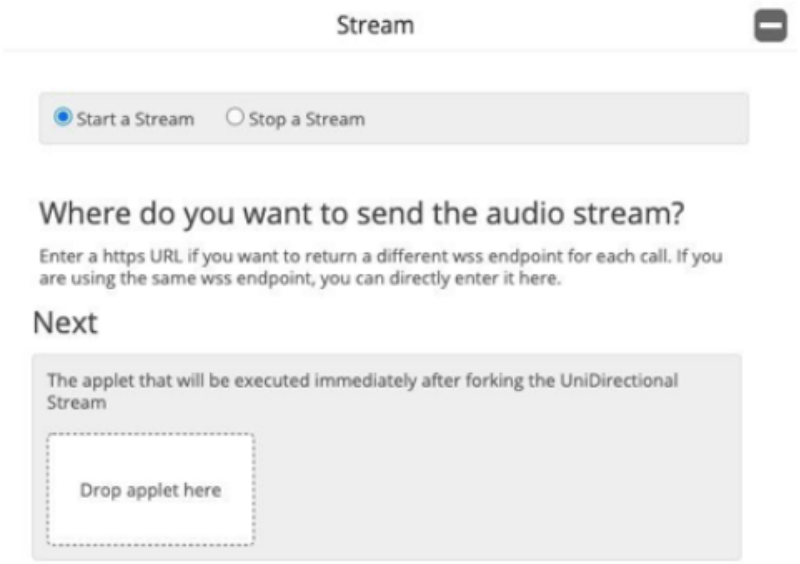
2.1.1. Stream Applet

Unidirectional Streaming

Unidirectional Streams allow you to Stream live calls over WebSocket in a single direction, from Exotel to the WebSocket Endpoint. Some of the use cases for this are live transcription, realtime monitoring of agents, realtime coaching etc.

Configuring a Unidirectional Stream-Stream applet

You can enable unidirectional streaming on a call flow using the Streaming Applet.



The applet takes 3 parameters :

1. Action - You can either start a new stream or stop a stream that you started earlier in the same call flow. You will use the Stop action if you have started a unidirectional Stream earlier in the same flow. When you choose stop, that is the only input you need to configure

2. URL - This is the URL to which Exotel will stream the voice media. You can either specify a WSS endpoint or an HTTPS endpoint. If you specify an HTTP/HTTPS endpoint, Exotel expects the HTTPS endpoint to return a WSS URL in its response. This is to allow

a. Dynamic endpoints for the same call flow
b. Have dynamic custom parameters that can be passed to the websocket endpoint to handle any application specific customization

When you specify a https endpoint, it must return a json with the key “url”

JSON

```
{
"url" : "wss://streamhandler.yourdomain.com"
}
```

On receiving this, Exotel will initiate a connection to wss://streamhandler.yourdomain.com

3. Next Applet

In the case if Unidirectional Stream, the Stream would be created and the call flow proceeds to the next applet configured

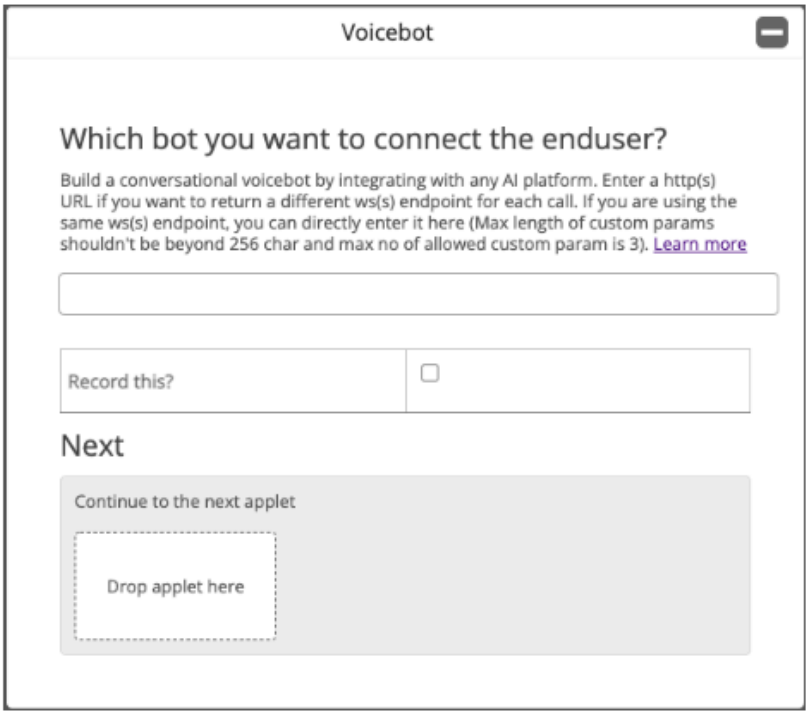
2.1.2. VoiceBot Applet

Configuring a Bidirectional Stream- Voicebot Applet

Bidirectional Streaming

Bidirectional Streams allow two-way flow of voice data over a websocket. Exotel would send the voice data of the caller to a websocket endpoint. The endpoint can return back voice data back on the websocket and Exotel would play it out to the caller. The primary use case for this is to enable building intelligent conversational bots that will help you optimize your workforce WebSocket

You can enable bidirectional streaming on a call flow using the Voicebot Applet.



The applet takes 6 parameters:

1. URL- This is the URL to which Exotel will stream the voice media. You can either specify a wss endpoint or a https endpoint. If you specify a http/https endpoint, Exotel expects the https endpoint to return a wss url in its response. This is to allow

a. Dynamic endpoints for the same call flow
b. Have dynamic custom parameters that can be passed to the websocket endpoint to handle any application-specific customisation.

There are two ways to put this:
a. Static method: ws(s) endpoint can be entered here, but it will remain the same for every call that you are going to make using this flow. eg: `ws://127.0.0.1:5001/media`
b. Dynamic method: We can enter http(s) URL which can return different ws(s) endpoints based on the use case.
eg: `https://yourdomain.com` this URL must return a ws(s) endpoint

Example

Exotel calls your URL with a GET request. For example, hitting:

```
https://yourdomain.com/resolve-endpoint
```

returns:

```
{"url":"wss://your-ws-handler.yourdomain.com/session/12345"}
```

Query string your endpoint will receive

Along with the request, Exotel appends call details as query parameters onto your URL. For example, if your endpoint is:

`https://yourdomain.com/resolve-endpoint`

Exotel will call it as something like:

```
https://yourdomain.com/resolve-endpoint?
CallSid=abc123xyz&CallFrom=0741XXXXXXX&CallTo=0804XXXXXXX&Direction=incoming&Created=Tue%2C+01+Sep+2026+14%3A45%3A29&DialWhomNumber=&From=0741XXXXXXX&To=0804XXXXXXX&CurrentTime=2026-09-01+14%3A45%3A29
```

Adding your own custom parameters

If you want to pass your own data through (a campaign ID, an agent ID, etc.), append it to your configured URL as a query parameter:

```
https://yourdomain.com/resolve-endpoint?CampaignId=campaign12345
```

Exotel appends the call details after your existing parameters, so you'll receive both together:

```
https://yourdomain.com/resolve-endpoint?CampaignId=campaign12345&CallSid=abc123xyz&CallFrom=...&CurrentTime=...
```

Note: this has been observed to work reliably for straightforward key=value parameters. With some parameter names, the returned key and value have been seen concatenated into a single malformed key with an empty value (paramName=value coming back as paramName:value=) instead of surviving as paramName=value. Test your specific parameter names on a tool like webhook.site before relying on this in production.

Response format expected from your endpoint

Your endpoint must respond with HTTP 200, Content-Type: application/json, and a JSON body with a single key, url, whose value is the wss endpoint to use for this call:

```
{ "url": "wss://your-ws-handler.yourdomain.com" }
```

Query parameters sent to your endpoint

Parameter	Description
CallSid	Unique identifier of the call.
CallFrom	Outgoing call: number the call is made from. Incoming call: number the call is received from.
CallTo	Outgoing call: number being dialed out. Incoming call: number where the call landed.
Direction	incoming or outbound-dial.
Created	Timestamp when the call is created (yyyy-mm-dd hh:mm:ss).
DialWhomNumber	Number of the agent who was dialed last. Blank if no dial attempt has happened yet at the point this request fires.
From	Incoming call: caller's number. Outgoing call: number of the first leg.
To	Incoming call: the ExoPhone the call came into. Outgoing call: the number dialed.
CurrentTime	Current server time (yyyy-mm-dd hh:mm:ss), at the moment this request is sent.

2. Authentication Options for WSS Endpoints- When configuring your WSS endpoint for the Voicebot Applet, Exotel supports the following authentication methods:

IP Whitelisting (Basic Setup)

Secure your WSS connections by whitelisting Exotel's outbound IP ranges. This avoids the need for credential exchange.

Reach out to hello@exotel.com to get the range of IPs.

Basic Authentication

Developers specify credentials in the WSS URL, but Exotel transmits them securely in headers during the connection.

- **WSS URL Configuration (Developer Input):**

```
wss://<API_KEY>:<API_TOKEN>@stream.yourdomain.com/<stream-path>
```

- **What Exotel Sends (Header):**

```
Authorization: Basic base64 (API_KEY:API_TOKEN)
```

This approach ensures compatibility and prevents credentials from being exposed in transit URLs.

3. WSS Sample Rate Parameters-Your WSS endpoint should support configurable sample rates. Exotel's Voicebot Applet can define the required rate as a query parameter.

- **8 kHz (Standard PSTN Quality – Default):**
  - wss://your-domain.com/?sample-rate=8000
- **16 kHz (Enhanced Quality):**
  - wss://your-domain.com/?sample-rate=16000
- **24 kHz (HD Quality):**
  - wss://your-domain.com/?sample-rate=24000

Default Behaviour: If no sample rate is explicitly defined, Exotel will use **8 kHz** as the default.

Best Practice: Use 16 kHz for most voicebot integrations (balanced quality vs. bandwidth). Use 8 kHz only when interworking with legacy PSTN and 24 kHz for HD audio experiences.

4. Custom parameters(Optional) - Custom params along with the endpoint. There are some validations that need to follow while providing custom params.

a. Maximum number of custom params that are allowed is 3.

b. Format of these params will like :

`ws://127.0.0.1:5001/media?param1=value1&param2=value2&param3=value3` (In Dynamic case, http(s) should return ws(s) URL in above format)

c. Total length of the params(bold part in above url) shouldn't be more than 256 characters.

5. Record - The checkbox gives an option to record the conversation and generate a recording URL available in passthru applet after voicebot applet.

6. Next Applet - In the case of a Bi-Directional Stream. The stream can end if the call is disconnected or the websocket is closed or the stream is explicitly stopped by the client. In the case of Bi-Directional Stream you do not need to add a explicit "Stop" Stream applet since the stream is automatically closed before executing the next Applet. The call flow proceeds to the next applet configured.

Video WalkThrough

You can find a quick walkthrough of a sample flow here.

Protocol

Communication between Exotel and customer endpoint happens over websocket connection.

Websocket messages - From Exotel

Each message in the websocket will be sent/received as a JSON string. Following are the types of messages that are sent:

- - Connected
- - Start
- - Media
- - DTMF
- - Stop
- - Mark (Only in Bidirectional)
- - Clear

Connected message:

After websocket connection is established, this message will be sent.

JSON

```
{
"event" : "connected",
}
```

Start message:

Start message will contain information about the stream parameters. It will be sent only once, right after the connected message. The custom parameters are picked from the URL configured in the Stream Applet. If you had mentioned the URL as

`wss://yourstream.service.com?queuename=premium&product=radio`

queuename and product would be passed in as keys with premium and radio as values.

JSON

```
{
"event" : "start",
"sequence_number" : 1,
"stream_sid" : "<stream sid>",
"start" : {
"stream_sid" : "<>",
"call_sid" : "",
"account_sid" : "",
"from" : "",
"to" : "",
"custom_parameters" : {
"Key1" : "value1",
"key2" : "value2"
},
"media_format" : {
"encoding" : "<>",
"sample_rate" : "<>",
“bit_rate” : “<>”
}
}}
```

Media message:

This message encapsulates the audio packets.

JSON

```
{
 "event" : "media",
"sequence_number" : 3,
"stream_sid" : "<stream sid>",
"media" : {
"chunk" : 2, "timestamp" : "10", "payload" : "<>"
}
```

media.chunk : chunk of the message
media.timestamp : Timestamp in milliseconds from the start of the stream.

Media in the payloads are sent in raw/slin (16-bit, 8kHz, mono PCM (little-endian)) encoded in base64. The same is expected from the client in the case of bi-directional streams (In the case of a Voice Bot) to be played back to the human.

DTMF message:

DTMF message is sent when the digits are pressed by the user once the connection with websocket is established. This is supported only for bidirectional streaming in voicebot applet.

JSON

```
{
"event" : "dtmf",
"sequence_number" : 1,
"stream_sid" : "<stream sid>",
"dtmf" : {
"duration" : "<duration in ms>”,  "digit": <>, }
}
```

Stop message:

Stop message is sent when the stream is stopped or the call has ended.

JSON

```
{
 "event" : "stop",
"sequence_number" : 10,
"stream_sid" : "<stream sid>",
"stop" : {
"call_sid" : “<>”,
"account_sid" : "<>",
 “reason” : “stopped or callended"
}
}
```

Mark Message:

Mark message is used only in bidirectional streaming to track media when it is completed.

JSON

```
{"event" : "mark",
"sequence_number" : 15,
"stream_sid" : "<stream sid>",
"mark" : {
"name" : "<label>”, }
}
```

Websocket messages - To Exotel

These messages will be used only in bidirectional streaming.

Mark Message:

Mark message is used only in bidirectional streaming to track media when it is completed. You can send a mark event message after sending a media message to request a notification when the audio that you have sent has been processed. You'll receive a mark event message with a matching name from Exotel when the audio is processed.

JSON

```
{"event" : "mark",
"sequence_number" : 15,
"stream_sid" : "<stream sid>",
"mark" : {
"name" : "<label>”, }
}
```

Media message:

This message encapsulates the audio packets.

JSON

```
{
 "event" : "media",
"sequence_number" : 3,
"stream_sid" : "<stream sid>",
"media" : {
"chunk" : 2, "timestamp" : "10", "payload" : "<>"
}
```

Clear message:

Clear message is used to clear the audio data that was sent before but not yet played. An example situation wherein it will be useful,

Developing human-like bots which can guess what he/she is going to say and send audio accordingly even before he/she completes it. When the guess goes wrong, we can clear that audio using a clear message.

```
{ "event": "clear", "stream_sid": "<stream sid>",}
```

Note: For Clear Event to work effectively, it is advisable to send media in smaller chunks. For instance, if two media messages of 5 seconds are sent to us, followed by Clear Event at the 3rd second, and the first message has already been picked up and played, the Clear event will only apply to the second media message. Therefore, sending media in smaller chunks can help prevent confusion and ensure that Clear Event works as intended.

Media Format

Media in the payloads are sent in raw/slin (16-bit, 8kHz, mono PCM (little-endian)) encoded in base64. The same is expected from the client in the case of bi directional streams to be played back to the caller.

Event Struct - Field Reference Table

Field Name	Type	JSON Key	Optional?	Description / Notes
Event	string	event	No	Type of the event: "start", "media", "stop", "dtmf", etc.
StreamSID	*string	stream_sid	Yes	Unique identifier for the stream session
SequenceNumber	*string	sequence_number	Yes	Ordering number for incoming media chunks
Start	*Start	start	Yes	Present when the event is a start event
Media	*Media	media	Yes	Present when the event is a media event (contains audio data)
Stop	*Stop	stop	Yes	Present when the event is a stop event
Mark	*Mark	mark	Yes	Present when the event is a mark event
Dtmf	*Dtmf	dtmf	Yes	Present when the event is a dtmf (key press) event

Sample Code

https://github.com/exotel/Agent-Stream and

https://github.com/exotel/Agent-Stream-echobot

Simulator to make streaming calls with dummy bot

https://github.com/exotel/voice-streaming/commit/d4696f3cb13fb5d75ac46bed2aaaa2afababa10f#diff-217ed82f87b78b399e304b07a159b27dff327c0f83adf4a2fc30b03bcbf84b01

Chunk size window for Bidirectional streaming use case:

Minimum chunk size: 3.2k [100ms data]

Maximum chunk size: 100k

Chunk size should always be in multiple of 320 bytes.

1. If the size is less than the minimum size, we may face audio issues due to network jitters.

2. If the size is greater than 100k, it might result in timeouts

3. if the size X [for ex 4096] which is not in multiple of 320 Bytes: In this case, the last packet will be of lesser size than 320 Bytes, & platform will wait for 20ms before sending next chunk. i.e. sending less amount of data than wait time, then this might result into audio gaps in between.

Limitations

1) When you start a Unidirectional Stream, the stream is forked immediately. In case you have a Connect applet post this, the audio stream, even when Exotel dials out multiple agent,s would be relayed (ringing). The client will have to handle this to filter only relevant parts of the stream. This limitation will be fixed in future releases

2) The number of Custom Parameters that can be passed in the START message is 3

3) The stream is sent as a mono-channel raw audio format, and the client will have to handle speaker diarization

If you have any questions or concerns, please connect with us using the chat widget on your Exotel Dashboard or WhatsApp us on 08088919888.

2.2. Passthru Applet

Overview

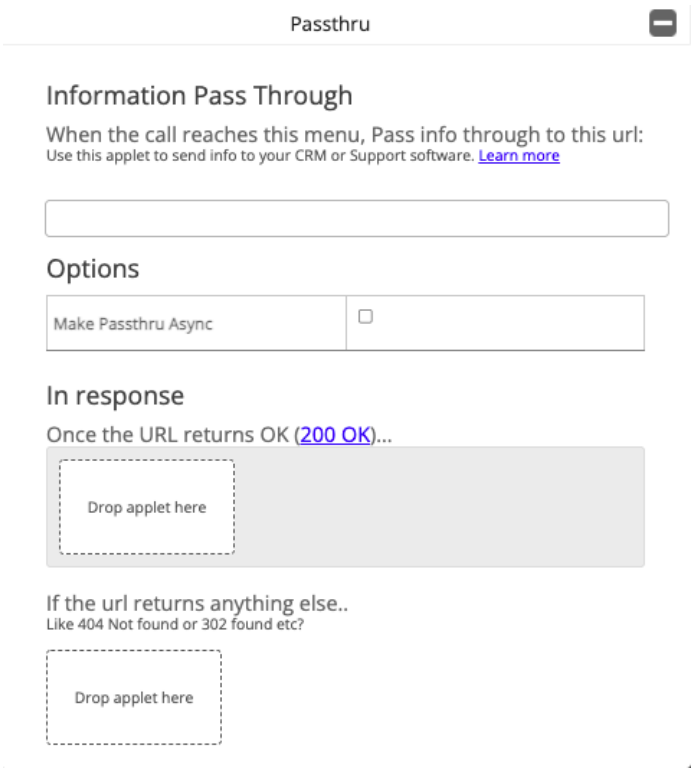
The Passthru Applet sends call and streaming metadata from Voice Applet flows to your server, especially useful when executing:

- **Voicebot Applet (Bidirectional Streaming)**
- **Stream Applet (Unidirectional Streaming)**

Passthru sends additional stream-specific parameters as query parameters, enriching standard passthru behaviour.

Refer to the standard passthru documentation here: Working with Passthru Applet

This guide is an **extended version** with additional parameters for streaming flows.



How It Works

Passthru makes an HTTP GET request to your URL, passing URL-encoded call and stream metadata.

Example:

HTTP

```
GET https://yourserver.com/passthru-handler?
CallSid=56b1234567abcdef89abcdef12345678&
flow_id=voicebot-flow-xyz123&
Direction=inbound&
From=+918888000000&
To=+912233344455&
Stream[StreamSID]=6f048d2e897a0d2d4029560f3f541947&
Stream[Status]=completed&
Stream[Duration]=28&
Stream[RecordingUrl]=https://recordings.exotel.com/exotelrecordings/exotelt/6f048d2e897a0d2d4029560f3f541947.mp3&
Stream[StreamUrl]=wss://stream.customer.com/media&
Stream[DisconnectedBy]=bot
```

Your backend must parse:

- URL query parameters
- Nested keys (e.g., Stream[Status])
- JSON strings

JSON Format (Raw Query String)

Sometimes, all parameters are sent in a single JSON string under the Stream key:

JSON

```
Stream={"StreamSID":"62xxx...",
"RecordingUrl":"https://recordings.exotel.com/...",
"Status":"completed",
"Duration":"28",
"StreamUrl":"ws://10.32.76.120:5007/media",
"DisconnectedBy":"user"}
```

- Ensure your backend deserialises this string correctly.

Handling Passthru Responses

Passthru requests can be handled synchronously or asynchronously:

- **Sync**: Exotel will immediately pass the call details synchronously during the call flow execution. It is possible only to make a binary decision using Passthru.
  - `200 OK → Option A`
  - `302 Found → Option B`
  - The person on call will wait until your URL responds.
- **Async**: Exotel will asynchronously pass the call details without interrupting the call flow execution. Use this mode when you don't want to dynamically change flow execution. The user will not experience a delay.

Streaming-Specific Fields (Flat Format)

Parameter	Description
Stream[StreamSID]	Unique streaming session ID
Stream[Status]	Status (completed, failed, cancelled)
Stream[Duration]	Stream duration (seconds)
Stream[StreamUrl]	WebSocket URL for streaming
Stream[RecordingUrl]	URL to recording (if enabled)
Stream[DisconnectedBy]	Disconnector (user, bot, NA)
Stream[DetailedStatus]	Present on failure (e.g., Streaming_call_throttled)
Stream[Error]	Error string (e.g., network failure or WS rejection)

Note: The above fields are added on top of standard passthru parameters.

Throttling Scenario

Concurrency limit breaches trigger:

JSON

```
Stream[Status]=failed
Stream[DetailedStatus]=Streaming_call_throttled
```

Implement retry or fallback strategies accordingly.

Failure Handling

Below are the most common failure and disconnection scenarios captured in the passthru payload:

Test 1: Call hung up during greeting

Expected: Capture who disconnected and status

JSON

```
{
"Status": "cancelled",
"DisconnectedBy": "user"
}
```

Test 2: Invalid WebSocket URL

Expected: Failure metadata including error, stream URL, and disconnector

JSON


```
{
"StreamSID": "b68e46bca0fdexxxxx89643f6d81196d",
"RecordingUrl": "https://recordings.exotel.com/exotelrecordings/exotelt/b68e46bca0fdexxxxx89643f6d81196d.mp3",
"Status": "failed",
"Duration": "0",
"StreamUrl": "ws://xx.x.xx.xxx:50/media",
"Error": "3009 failed to establish ws conn dial tcp 10.1.76.120:50: connect: connection refused",
"DisconnectedBy": "NA"
}
```


Test 3: Call cancelled after delay in streaming setup

Expected: Accurate cancellation reason and disconnection origin

JSON


```
{
"StreamSID": "6f048d2e897a0dxxxxx9560f3f541947",
"Status": "cancelled",
"DisconnectedBy": "user"
}
```


Test 4: Call disconnected by bot after successful conversation

Expected: Final stream status and session metadata

JSON


```
{
"StreamSID": "f5dd49d7ac3xxxxx4b491ce948501947",
"Status": "completed",
"Duration": "29",
"StreamUrl": "wss://e33e-54-251-51-3.xxxxx-free.app",
"DisconnectedBy": "bot"
}
```


These structured outputs ensure consistent understanding of streaming lifecycle and failure context.

Note: DisconnectedBy values are standardized to: user, bot, NA.

Passthru logs enriched with detailed Error messages improve debugging and recovery mechanisms.

2.2.1. More on Passthru Applet

Best Practices

- Position Passthru immediately after Stream or Voicebot Applets.
- Backend should handle both flat and JSON query formats.
- Trigger fallback based on Stream[DetailedStatus].
- Actively monitor concurrency limits.
- Log StreamSID, RecordingUrl, and detailed errors for debugging and auditing.
- Clearly interpret and handle disconnect causes (DisconnectedBy).
- Passthru backend responses must comply with synchronous (HTTP codes) or asynchronous processing models appropriately.

Use Cases

- Real-time monitoring of stream health
- Dynamic routing based on real-time stream conditions
- Enhanced debugging through detailed error reporting
- Optimised concurrency and call management
- Graceful error handling and escalation mechanisms

Deployment Strategy

Follow structured rollout:

1. **Dev Environment**: Validate passthru logic with mock logs.
2. **QA/Staging**: Test real call scenarios thoroughly.
3. **Canary Release (Prod)**: Roll out feature gradually, monitoring closely.
4. **Full Release**: Expand to all users upon successful canary release.

Ensure rollback capabilities via feature flags and previous stable versions.

Summary

The Passthru Applet for AgentStream significantly enhances standard passthru functionality by providing extensive stream metadata, failure reasons, and real-time observability. Leverage these capabilities to:

- Monitor and optimize streaming performance
- Facilitate precise error handling and debugging
- Enable intelligent and dynamic routing decisions

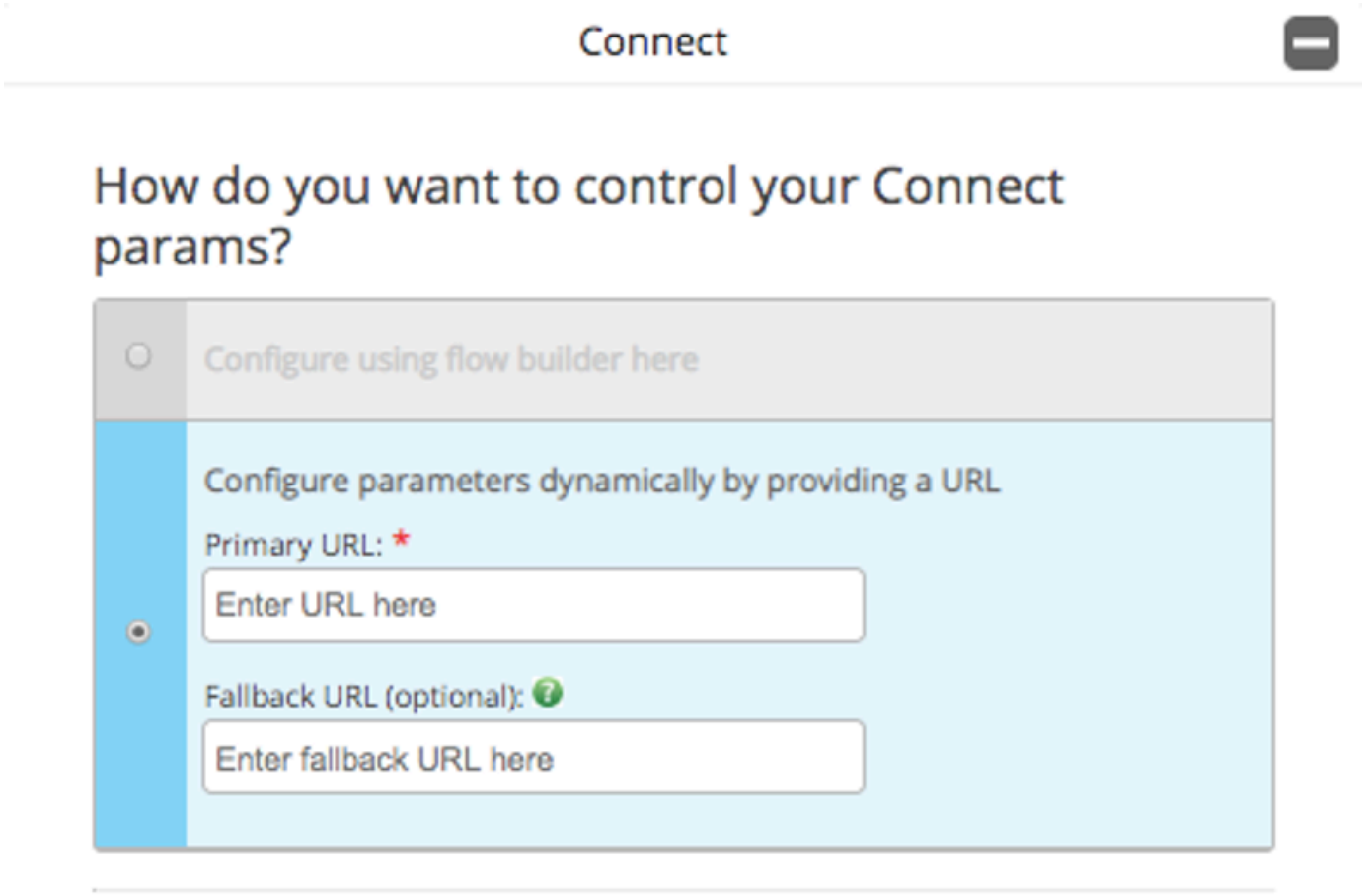
Implement these advanced passthru mechanisms to build robust, observable, and scalable voice-streaming integrations with Exotel APIs.

2.3. Connect Applet for Agent handover

Connect Applet: Configuration, Dynamic URL, and Escalation via Passthru for Agent handover

Overview

You can configure the Connect applet's parameters in one of two ways:



1. **Flow Builder** – set parameters directly in the applet's UI.
2. **Dynamic URL** – control parameters at runtime via your own application endpoint.

Regardless of which method you choose, you must still configure the **transitions** (which applet runs next) while building the flow.

If you configure parameters dynamically, you set:

- A **Primary URL**, which handles the requests.
- An optional **Fallback URL**, contacted if something goes wrong with the Primary URL.

Request (Exotel → Your Application)

If an application URL is set for the Connect applet, Exotel makes a **GET** request to that URL with call details as URL-encoded HTTP query parameters. Only some parameters may be passed, depending on where the Connect applet sits in the flow.

Header

Header	Description
Exotel-Version	Version of Connect applet parameters against which your endpoint's response will be validated. Current version: 1.0.

Query Parameters

Parameter	Description
CallSid	Unique identifier of the call.
CallFrom	Outgoing call: number the call is made from. Incoming call: number the call is received from.
CallTo	Outgoing call: number being dialed out. Incoming call: number where the call landed.
Direction	incoming or outbound-dial.
Created	Timestamp when the call is created (yyyy-mm-dd hh:mm:ss).
DialCallDuration	Seconds from when the call is triggered to when the second leg ends (including conversation time). Can be 0 depending on the previous applet and if there's no second leg.
StartTime	Timestamp when the call started (yyyy-mm-dd hh:mm:ss).
EndTime	Constant value 1970-01-01 05:30:00 (Unix epoch). Use the Call Details API a few minutes after the call ends to get accurate end time.
CallType	See table below.
DialWhomNumber	Number of the agent who was dialed last.
flow_id	Flow ID associated with the call.
From	Incoming call: caller's number. Outgoing call: number of the first leg.
To	Incoming call: the ExoPhone the call came into. Outgoing call: the number dialed.
CurrentTime	Current server time (yyyy-mm-dd hh:mm:ss).

CallType values:

Scenario	Value
IVR only, no Connect applet	call-attempt
Call conversation happened	completed
Client hung up during Connect applet	client-hangup
Connect applet, no agent picked up	incomplete
Went to voicemail applet	voicemail

Conditional Parameters

Passed only if certain conditions are met:

Parameter	Description
DialCallStatus	What happened on the second leg if the previous applet was "Connect". Values: completed, busy, no-answer, failed, canceled.
digits	Passed if a Gather or IVR applet preceded this one; equals the digits entered. Note: comes wrapped in double quotes (") – trim them to get the actual digits.
CustomField	If the call was initiated via API, the value passed as CustomField in that API call.
RecordingUrl	Populated if the previous applet was "voicemail"; contains the voicemail recording URL. There may be a delay before it's accessible, depending on recording length.

Response (Your Application → Exotel)

This is what Exotel expects back from your GET request. It decides how Connect executes during the call.

Response header: Content-Type: application/json

Sample response

JSON

```
{
  "fetch_after_attempt": false,
  "destination": {
    "numbers": ["+919812345678"]
  },
  "outgoing_phone_number": "+918047115777",
  "record": true,
  "recording_channels": "dual",
  "max_ringing_duration": 45,
  "max_conversation_duration": 3600,
  "music_on_hold": {
    "type": "operator_tone"
  },
  "start_call_playback": {
    "playback_to": "both",
    "type": "text",
    "value": "This text would be spoken out to the callee"
  }
}
```

Response parameters

Parameter	Mandatory/Optional	Description
fetch_after_attempt	Optional; default false	Whether to re-fetch parameters (including destination numbers) after each unsuccessful dial attempt. false: dial happens based on the initial response only, no subsequent hits to the URL. true: Connect re-fetches parameters (hits the URL again) if a dial attempt fails. If two consecutive fetches return the exact same destination numbers, Exotel will not retry further even if this is true. Re-fetch request includes standard params plus <connect> params from previous dial attempts; response format is the same as above.
destination	Mandatory	The destination(s) to dial. See sub-parameters below.
destination.numbers	–	Array of E.164-format numbers to dial, in the order they appear. Example: "destination": {"numbers": ["+622131921111", "+622131921112"]}.
destination.trunk	–	Route the call to a SIP trunk instead of a number, e.g. "destination": {"trunk": "trunk-2134"}. Used for SIP Transfer to agents/contact centers (see Escalation via Passthru Applet below).
outgoing_phone_number	Optional; default = incoming ExoPhone	ExoPhone to dial out from (E.164 format). Must exist in your account. Subject to telecom circle/region restrictions (e.g., outgoing ExoPhone in Delhi can't be used if the incoming ExoPhone is in Bangalore). Both legs' ExoPhones can be on the same server. If the ExoPhone isn't in your account or fails validation, Exotel falls back to using the same ExoPhone as the first leg. Consult Exotel support before using this.
record	Optional; default false	Whether to record the call.
recording_channels	Optional; default single	single or dual (caller and callee in separate channels).
max_ringing_duration	Optional; default 30	Ringing duration limit, in seconds. Can be increased up to 60.
max_conversation_duration	Optional; default 900 (15 min)	Conversation duration limit, in seconds. Can be increased up to 4500 (75 min).
music_on_hold	Optional; default default_tone	default_tone (Exotel's default), operator_tone (audio returned by the operator on the dialing channel as-is), or custom_tone (audio URL provided in the response). Example: {"type": "custom_tone", "value": "<audio_url>"}.
parallel_ringing	Optional	Dial all destination numbers simultaneously: {"activate": true, "max_parallel_attempts": 5}. max_parallel_attempts range 1–10, default 5. Chargeable feature – confirm with your Account Manager or hello@exotel.in before use.

Parameter	Mandatory/Optional	Description
dial_passthru_event_url	Optional	URL requested for dial start and dial end events.
start_call_playback	Optional	Play a recording or TTS to the called number. playback_to: both or callee. type: audio_url or text. Example: {"playback_to": "both", "type": "text", "value": "hello, this is a sample text"}. Audio file requirements: 8 kHz sample rate, 16-bit depth, 128 kbps bit rate, mono channel, .wav format. If reusing the same audio_url filename, it will be cached by Exotel servers – use dynamic filenames if the audio content changes each time.
custom_params	Optional	Custom SIP headers passed with a trunk-routed call, e.g. "custom_params": "param1=value1&param2=value2". Up to 3 custom headers, max 200 bytes total. Headers prefixed with Exotel- or Veeno- are platform-reserved.

All of the above can also be set via the dashboard if you configure the Connect applet through the Flow Builder instead of a Dynamic URL.

Fallback URL triggers

The Fallback URL is called when:

- The Primary URL doesn't return HTTP 200 OK.
- The Primary URL doesn't respond within the timeout period (5 seconds).
- The Primary URL returns an invalid response:
  - Content-Type is not application/json.
  - Mandatory parameters are missing.
  - audio_url / text isn't a valid HTTP/HTTPS URL returning 200.

Transition to Next Applet

Set these transitions in the Flow Builder to decide what runs next:

Scenario	Behavior
After the call conversation ends	Triggered if a conversation occurred.
If nobody answers	Triggered if the number was dialed but no conversation occurred. Falls back to "We didn't dial anyone" if not set.
We didn't dial anyone	Triggered if no dial occurred at all, including: both Primary and Fallback URLs timed out or returned a non-2xx response; both URLs returned an invalid payload; the returned number was in an invalid format; or an empty destination was returned.

Notify or Escalate via Passthru Applet

Use the **Passthru Applet** to notify your contact center or escalate a call to an agent. Two modes are supported:

1. **Notification Mode** – the bot triggers a webhook via the Passthru Applet.
2. **Transfer Mode** – the bot requests a SIP Transfer to hand the call to an agent, by creating a trunk in Exotel and routing the Connect applet's destination to it (using destination.trunk, above).

Setting up SIP Transfer

1. Follow the SIP Trunking API Reference to create a trunk for inbound calls.
2. Create a flow using the **Connect Applet** in App Bazaar.
3. In the **Dial Whom** field, use sip:<TrunkID>.
4. For custom routing, use a **Dynamic URL** to fetch the destination URI and pass headers, as described above.

- Supports **up to 10 custom SIP headers**, e.g.,X-*=value, X-param1=value1 (max 800 bytes each)- max 4000 bytes total)
- Headers prefixed with Exotel- or Veeno- are platform-reserved

Connect Applet – Dynamic URL response example for SIP Transfer:

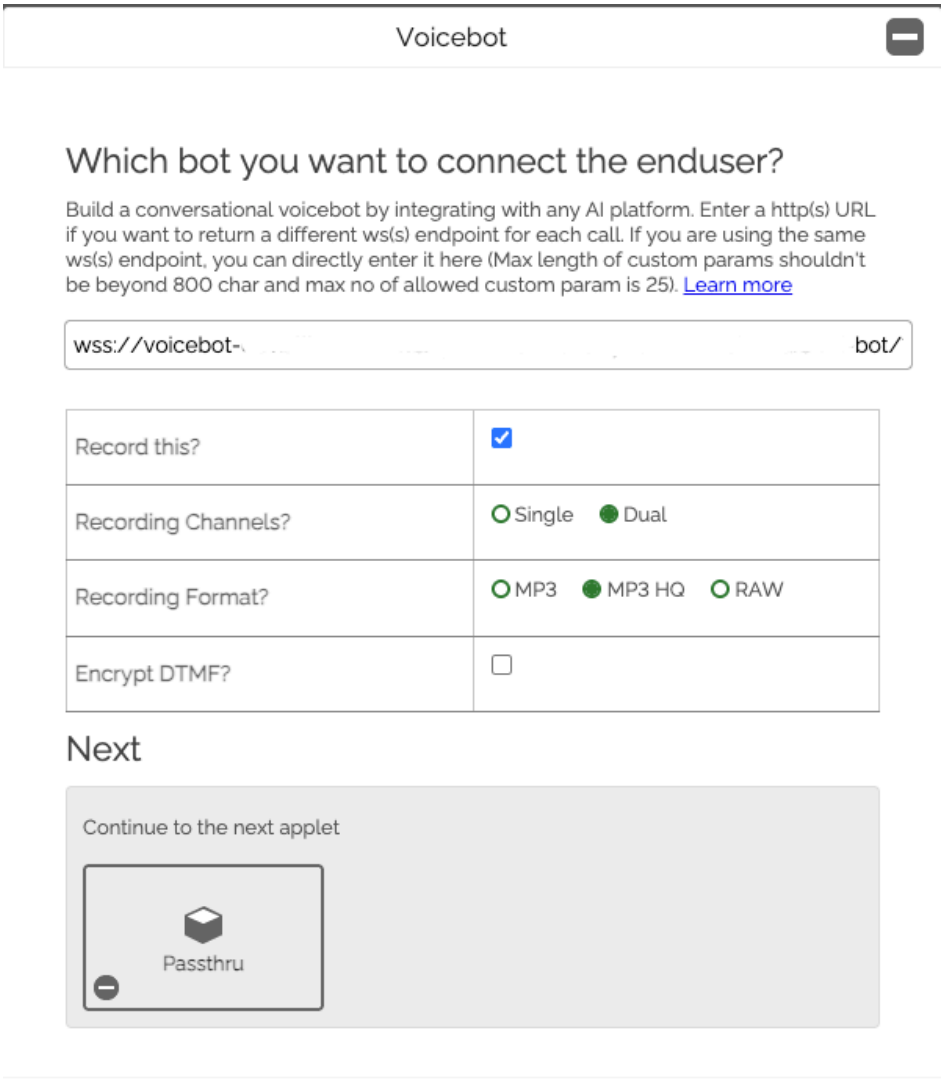
JSON

```
{
  "fetch_after_attempt": false,
  "destination": { "trunk": "trunk-2134" },
  "custom_params": "param1=value1&param2=value2",
  "record": true,
  "recording_channels": "dual"
}
```

References

- **Programmable Connect: Working with Connect Applet (Dynamic URL)**
- **Passthru Applet**

2.4. Recording Options in the Voicebot Applet



Voicebot Applet now allows you to record live conversations between your bot and customers for quality monitoring, compliance, or training purposes.

You can configure recording behaviour directly within the applet under the **"Record this?"** option.

Common Use Cases:

- **Compliance Audits:** Maintain proof of consent and adherence to regulatory standards.
- **Quality Monitoring:** Evaluate bot performance, tone, and speech recognition accuracy.
- **Training and Optimization:** Use call recordings to retrain AI models and improve conversation design.
- **Dispute Resolution:** Retrieve exact customer-bot interactions for support and verification.

How to Enable Recording

1. Open your **Voicebot Applet** in App Builder.

2. In the **Advanced Settings** section, enable the toggle **Record this?**
3. Once enabled, you'll see two additional configuration fields:

Recording Channels

- **Single** – Captures a mono recording (both streams mixed).
- **Dual** – Captures each participant's audio on separate channels for detailed analysis.

Recording Format

- **MP3** – Standard quality (default, 8 kbps).
- **MP3 HQ** – High-quality stereo (32 kbps/channel).
- **RAW** – Uncompressed audio for advanced analytics or machine learning models.

Note: HQ and RAW recording options are available only for accounts where the **High-Quality Recording** feature is activated. If you don't see these options, please reach out to your Account Manager or email **hello@exotel.com**.

3. StreamKit

3.1. StreamKit Cloud

StreamKit Developer Quickstart Guide

Bridge your Contact Centre (SIP) and Voicebot (WSS) using Exotel StreamKit Cloud Connector

1. Overview

StreamKit is Exotel's cloud-based SIP ↔ WSS connector that allows you to connect any SIP-based contact centre (like Avaya, Genesys, Ameyo, or custom PBX) with any WSS-based AI platform (Dialogflow CX, OpenAI, AWS Lex, etc.) in real time.

This document will help you set up an end-to-end integration using:

- **Exotel Dynamic SIP Trunking APIs**
- **AgentStream Applets (Stream and Voicebot / Passthru)**
- **Custom SIP header and routing configuration**
- **Voicebot and Agent Assist integration flows**

2. Prerequisites

Requirement	Description
Exotel Account	Sign up for an Exotel account by selecting browser calling and get your `Account SID, Auth Key,` and `Auth Token.`
Virtual Number	Buy or provision a virtual number/Exophone (DID) from Exotel by reaching out to Hello@exotel.com and requesting for VoIP Exophone In case, you need the IP<>PSTN Intermix request for Veeno Exophone
SIP/IVR System	A contact centre or PBX that supports SIP INVITE routing.
Voicebot Endpoint	WSS server endpoint or AI provider URL (e.g., WebSocket URL:ws://127.0.0.1:5001/media).
IP Whitelisting	Your Contact Centre PBX/Server IPs must be whitelisted on Exotel.
Exotel Singaling and Media Ips	Your Contact Centre firewall must whitelist Exotel IPs and Ports
Test	Do test calls, routing, and in case any support is required, reach out to Exotel Phone System

3. Step-by-Step Setup

Step 1: Create an Account and Virtual Number

Signup on Exotel **Link** and select browser calling while signing up

Reach out to hello@exotel.com to procure the VoIP Exophone

Once done, note:

- **API Settings Link**
  - **Account SID**
  - **Subdomain**
  - **Auth Key**
  - **Auth Token**
- **Virtual Number/Exophone Link**

Step 2: StreamKit Cloud SIP trunking Setup

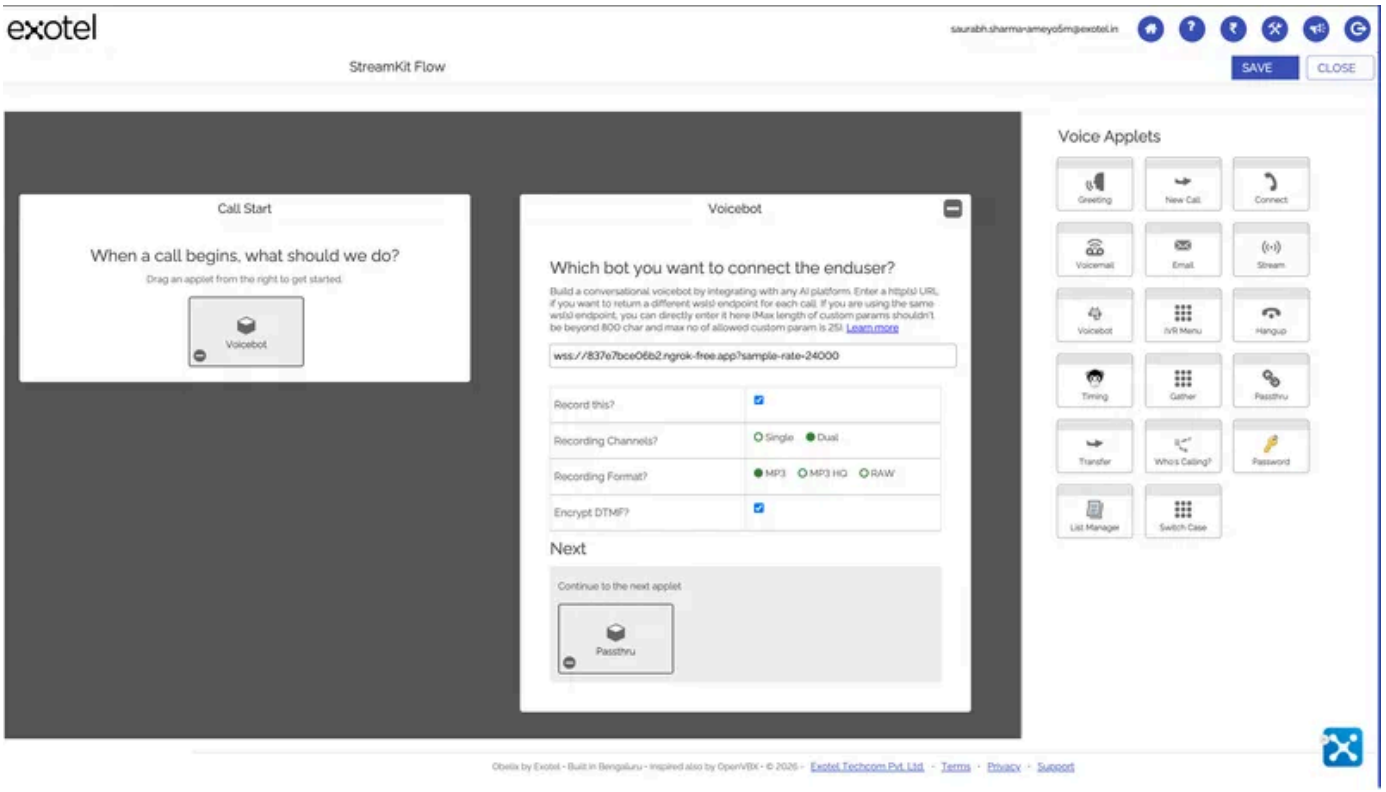
Whitelisting ensures Exotel can securely send/receive SIP traffic from your servers.

- You can add multiple IPs.
- for more details refer **Detailed SIP Trunking API Reference**

Step 3: Set Up Your Call Flow

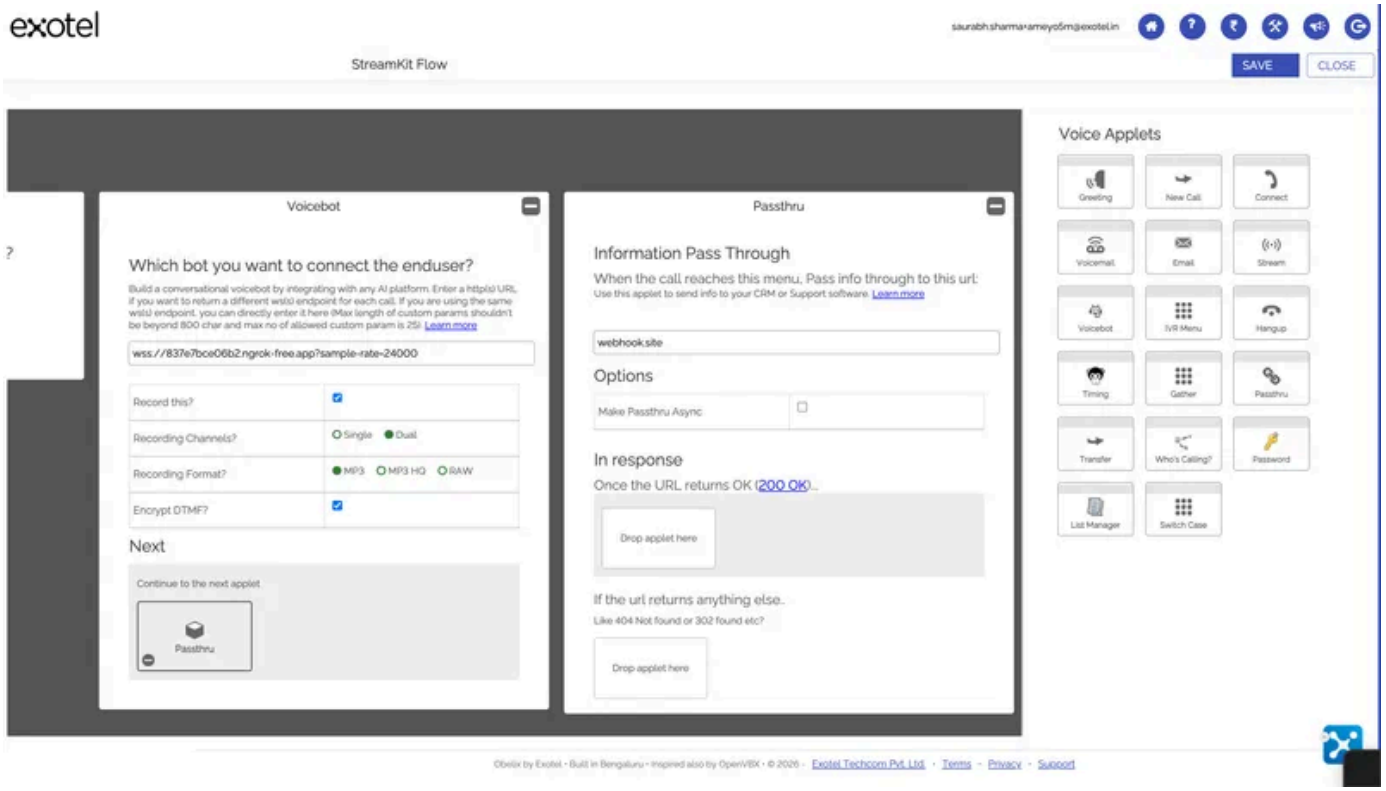
Use Exotel **Flow Builder** to define how the call moves from PSTN → SIP → WSS.

1. Log into Exotel Dashboard → **Create a Flow**
2. Add **Voicebot Applet** to connect SIP calls to your Voicebot.



Reference: **AgentStream Applets**

3. If you plan to integrate **Agent Assist**, use Stream Applet.
4. **Passthru Applet**.



Reference: **Passthru Applet**

5. **Programmable Connect: Working with Connect Applet (Dynamic URL)** in case you want to route a call back to the contact centre
6. We have to add the greeting applet as the first applet with text as ...
    - Note: if we don't add this, in that case, we will observe the voice blank issue, and the call will get disconnected in around 20 to 23 seconds
7. Map VoIP Exophone to the flow **Using Existing Phone Number** (for VoIP exophone procurement reachout to support)

Step 4: Send SIP INVITE at the Right Stage

From your Contact Centre or PBX, trigger a **SIP INVITE** to the Exotel SIP domain with required headers. Please do followthe mandatory headers as per the setup guide

Refer aswell **Network & Firewall Configuration**

Use this flow:

```
Customer → Contact Center (SIP PBX) → SIP INVITE → Exotel StreamKit → Voicebot (WSS)
```

Please do send X-Exotel-AccountSid:(Account_SID) for correct routing and CLI capture- Mandate

Step 5: Add Custom Headers

In SIP invite Contact center can pass custom headers ,

Custom headers help StreamKit voicebot applet decide which bot or campaign to route the call to.

Sample headers supported:

```
X-Custom-Header: SalesBot
```

```
X-UUI: XXXXXX
```

Supports upto 45 bytes each

{Coming Soon}

```
X-Custom-Header: SalesBot
```

```
X-UUI: XXXXXX
```

```
X-User-to-User: SalesBot
```

```
X-Campaign-ID: XXXXXX
```

```
X-Customer-ID: SalesBot
```

```
X-JWT: XXXXXX
```

```
X-Cert-Obj: XXXXXX
```

Supports 800 bytes per header and overall 4KB total

Note: Currently, additional custom headers above are supported on VoIP Exophone only.

Step 6: Talk to the Voicebot (WSS)

Once the SIP INVITE is accepted:

- StreamKit opens a WSS connection to your voicebot.
- The SIP custom headers(X-UUI and X-Custom-Headers) are supported in two ways
  - It is received as part of the start event custom_parameters of the Voicebot/Stream Applet
- **custom_parameters** Key1 value is fetched from wss query params and passed at start event

```
Start Message received: {"event":"start","stream_sid":"b6c581dXXXXXX67a2199q","sequence_number":"1","start":
{"stream_sid":"b6c581xxxxxxxxxbfe027a2199q","call_sid":"2449xxxxx1a64fbxxxx199q","account_sid":"Tenant-
ID","from":"sip:0806xxxx509","to":"015xxxx0321","custom_parameters":{"Custom-
Header":"Salesbot","UUI":"XXXXXXX","key1":"value1"},"media_format":
{"encoding":"base64","sample_rate":"8000","bit_rate":"128kbps"}
```

{Coming Soon}


```
Start Message received: {"event":"start","stream_sid":"b6c581dXXXXX67a2199q","sequence_number":"1","start":
{"stream_sid":"b6c581xxxxxxxxxbfe027a2199q","call_sid":"2449xxxxx1a64fbxxxx199q","account_sid":"Tenant-
ID","from":"sip:0806xxxx509","to":"015xxxx0321","custom_parameters":{"Custom-
Header":"Salesbot","UUI":"XXXXXXX","User-to-User":"XXXXXXX","Campaign-ID":"XXXXXXX","Customer-
ID":"XXXXXXX","JWT":"XXXXXXX","Cert-Obj":"XXXXXXX","key1":"value1"},"media_format":
{"encoding":"base64","sample_rate":"8000","bit_rate":"128kbps"}
```


- Received as additional SIPCustomHeader Query parameters in https `GET` request of programmable applets: Passthru and Connect.

Sample:

```
"SIPCustomHeader": {
    "Custom-Header": "SalesBot",
    "UUI": "XXXXXXX"
```

{Coming Soon}

- Received as additional SIPCustomHeader Query parameters in https `GET` request of programmable applets: Passthru ,Voicebot, Stream and Connect.

JS GET

```
GET /your-voicebot-url?CallSid={call_sid}
    &CallFrom=073XXXXXXX
    &CallTo={destination}
    &Direction=incoming
    &SIPCustomHeader[Custom-Header]=salesbot
    &SIPCustomHeader[UUI]=handoff-token-up-to-800-bytes
    &SIPCustomHeader[User-to-User]=XXXXXXXXXXXX...
    &SIPCustomHeader[Campaign-ID]=outbound-camp-42
    &SIPCustomHeader[Customer-ID]=CUST-556677
    &SIPCustomHeader[JWT]=eyJhbGciOiJIUzI1NiJ9...
    &SIPCustomHeader[Cert-Obj]=eyJhbGciOiJIUzI1NiJ9...
```

- RTP ↔ PCM16 media is exchanged.
- Bot starts real-time ASR/TTS or speech-to-speech conversation.

Reference Doc: **Voicebot Applet and bidirectional Integration Guide**

Step 7: Notify or Escalate via Passthru Applet

Use **Passthru Applet** to notify your contact center or escalate to an agent.

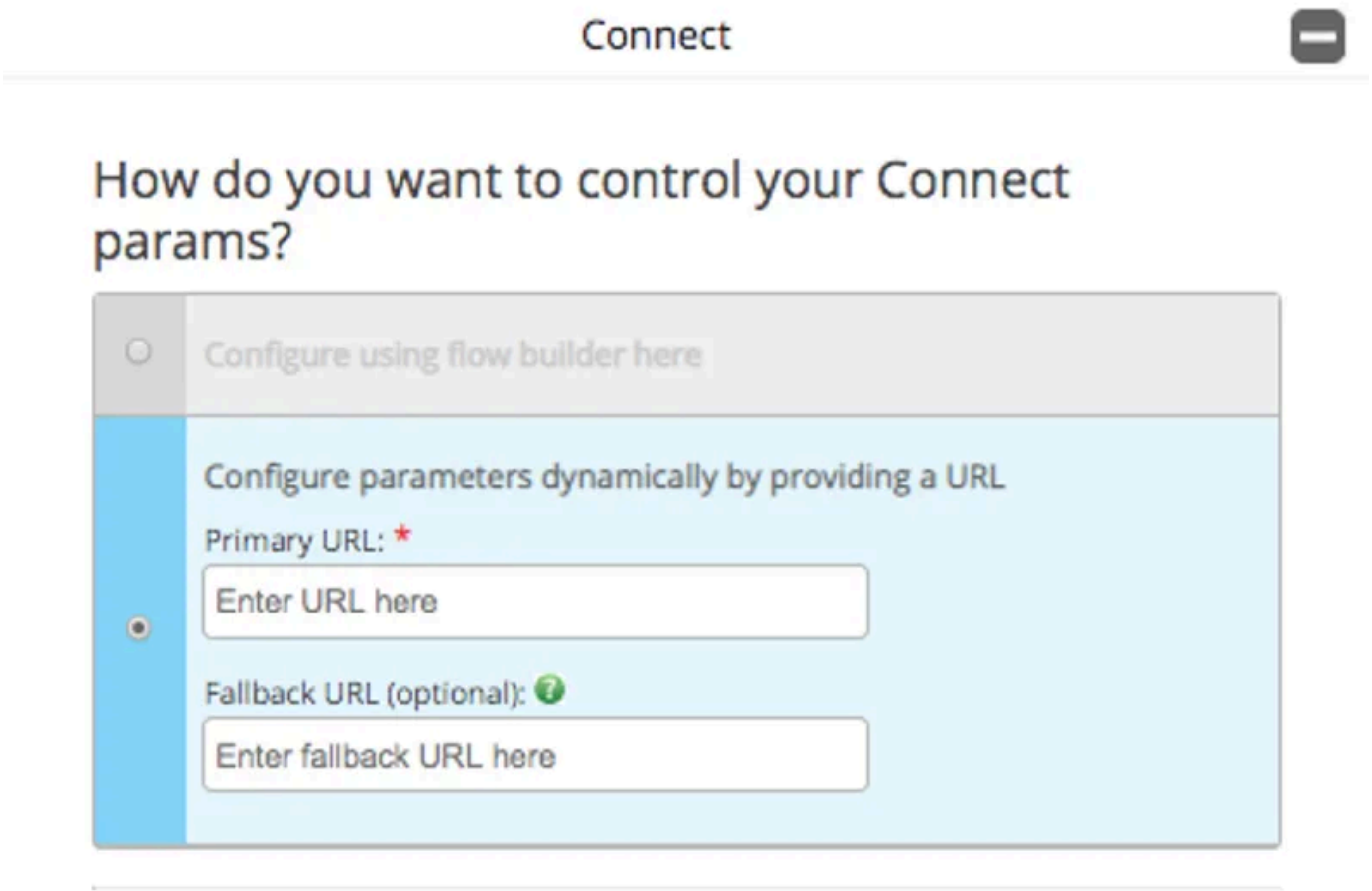
Supported modes:

1. **Notification Mode:** Bot triggers webhook via **Passthru Applet.**

2. **Transfer Mode:** Bot requests SIP Transfer to transfer call to agent by creating another trunk at Exotel

API Steps to create a Trunk

- **Detailed SIP Trunking API Reference** for Inbound call
- Create a Flow using **Connect Applet** in App Bazaar
- Use `sip:<TrunkID>` in the **Dial Whom** field
- For custom routing, use a Dynamic URL to fetch the destination URI and pass headers



- Supports **up to 3 custom SIP headers**, e.g., X-param1=value1 (max 200 bytes total)
- Headers prefixed with Exotel- or Veeno- are platform-reserved

{Coming Soon}

- Supports **up to 10 custom SIP headers**, e.g.,X-*=value, X-param1=value1 (max 800 bytes each)- max 4000 bytes total
- Headers prefixed with Exotel- or Veeno- are platform-reserved

Connect Applet – Dynamic URL Response Example:

JSON

```
{
"fetch_after_attempt": false,
"destination": { "trunk": "trunk-2134" },
"custom_params": "param1=value1&param2=value2",
"record": true,
"recording_channels": "dual"
}
```

Reference: **Programmable Connect: Working with Connect Applet (Dynamic URL)**

Docs:

Passthru Applet (AgentStream)

4. Special Scenarios

Scenario	Description
Voicebot Mode	SIP ↔ WSS full-duplex. Bot handles ASR + TTS or speech-to-speech using Voicebot applet
Agent Assist Mode	Unidirectional WSS stream from caller to AI (for analysis, not speech output) using Stream start and stop applet
Agent Monitored Bot	Hybrid – bot monitors conversation and provides insights.
Post-Call Notify	StreamKit triggers the Passthru Applet after call completion for analytics or CRM updates.

5. Example Flow

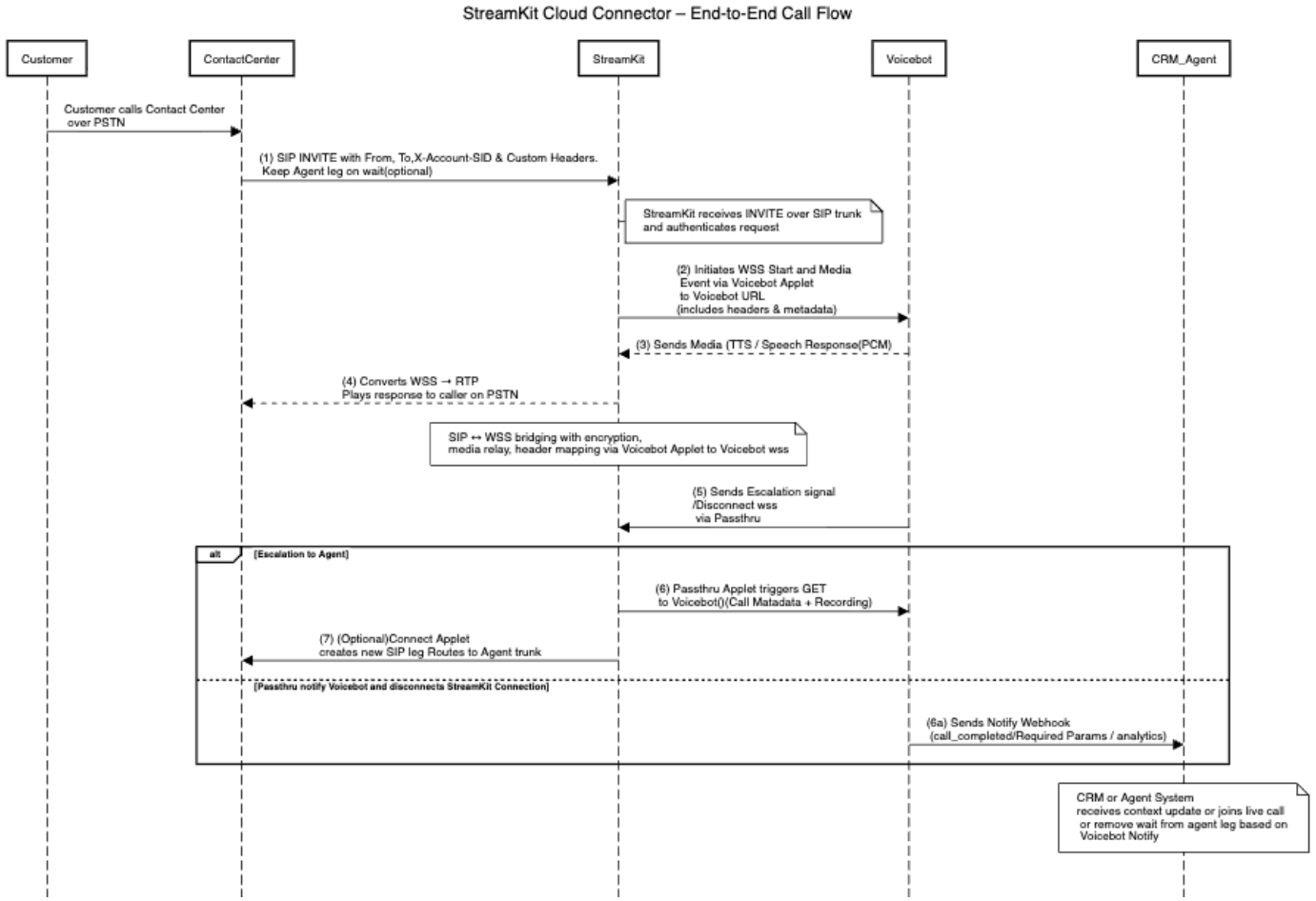
Voicebot Scenario

```
PSTN Call → CC (SIP PBX) → Exotel StreamKit → WSS Voicebot
```

Agent Assist Scenario

```
PSTN Call → CC → Exotel StreamKit (Passthru) → WSS Bot (listen only)
```

```
Bot → Notify API → CRM / Dashboard
```



6. Compliance & Data Localization

- All media and logs stay within India (Local PoPs).
- DoT and TRAI voice compliance is ready.
- Optional encrypted call recording with region-specific storage.

7. Quick Links

Resource	URL
Dynamic SIP Trunking	https://docs.exotel.com/dynamic-sip-trunking
Virtual SIP Trunk to Flow Guide	https://docs.exotel.com/dynamic-sip-trunking/exotel-virtual-sip-trunking-to-flow-integration-guide
AgentStream Applet	https://docs.exotel.com/exotel-agentstream/agentstream-applets
Passthru Applet	https://docs.exotel.com/exotel-agentstream/passthru-applet
Flow & API Config	https://docs.exotel.com/dynamic-sip-trunking/flow-and-api-configuration-guide-for-voice-ai-and-contact-centre-platforms

4. Connectors

4.1. Connect Voice AI with AgentStream

Use Exotel **AgentStream** to stream live call audio to your bot over WebSocket, and play bot audio back to the caller.

This page gets you from zero to a working outbound test call. Pick one provider guide when you are ready to wire a full voice agent.

API reference: Connect Voice AI API

How it works

Text

```
Caller  ⟷  Exotel  ←WSS PCM→  your bot (public wss://)
```

1. You run a bot that speaks the AgentStream events: `connected`, `start`, `media`, `stop` (and optionally `mark` / `clear`).
2. You expose that bot as a public `wss://` URL (TLS).
3. You place a call with Connect Voice AI (`StreamType=bidirectional`, `StreamUrl` = your bot).

Telephony default on AgentStream is **8 kHz, 16-bit PCM, mono**. Always pass the sample rate your bot expects on the StreamUrl query string when the bot documents it (usually `?sample-rate=8000`).

Before you start

You need	Notes
Exotel account	Account SID, API Key, API Token, ExoPhone (CallerId)
A phone number to call	E.164, for example `+91…`
Public WSS	cloudflared, ngrok, or your own HTTPS load balancer
Python 3.10+	For the sample bots and `place_connect_call.py`

Clone the sample repo:

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream
cp shared/env.exotel.example shared/.env.exotel
# Edit shared/.env.exotel with EXOTEL_ACCOUNT_SID, EXOTEL_API_KEY,
# EXOTEL_API_TOKEN, EXOTEL_CALLER_ID
```

Pick a provider (about 15 minutes each)

Guide	Mode	Default port	StreamUrl
ElevenLabs	Speech-to-speech (ConvAI)	10002	`wss://HOST/v1/convai/conversation/exotel`
OpenAI Realtime	Speech-to-speech (Realtime GA)	5000	`wss://HOST/?sample-rate=8000`
Sarvam	STT → TTS (Saaras / Bulbul)	8000	`wss://HOST/ws?sample-rate=8000`

Replace `HOST` with your tunnel hostname (no `https://` prefix on StreamUrl – use `wss://`).

Place a Connect call (shared helper)

From the repo root, after your bot and tunnel are up:

Bash

```
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_HOST/…see table above…"
```

Endpoint used by the helper:

`POST /v1/Accounts/{AccountSid}/Calls/connect`

Auth: HTTP Basic (API Key / API Token)

Field	Required	Description
`From`	yes	Callee in E.164
`CallerId`	yes	Your ExoPhone
`StreamUrl`	yes	Bot WSS URL (keep under ~600 characters)
`StreamType`	yes	`bidirectional`

Go-live checklist (any provider)

- Bot listens on a stable host with a valid TLS certificate (not a disposable laptop tunnel for production).
- Secrets live in environment variables or a secret store – never in git.
- StreamUrl path and `sample-rate` match the provider README exactly.
- You verified pitch (not slow/deep) and at least one full user turn on a real handset.
- You read **AgentStream: WSS errors and handling** for disconnect and timeout behaviour.

Next

- **ElevenLabs Conversational AI**
- **OpenAI Realtime**
- **Sarvam STT/TTS**

4.1.1. ElevenLabs's ElevenAgents -AgentStream Integration

Connect an Exotel phone call to an **ElevenLabs Conversational AI** agent over AgentStream. Audio stays speech-to-speech: Exotel PCM ↔ your bridge ↔ ElevenLabs ConvAI.

Sample code: `integrations/elevenlabs` in the **Agent-Stream** repo.

If you have not set up Connect yet, start with **Connect Voice AI with AgentStream**.

What you get

- Outbound (or inbound via applet) calls that talk to your ElevenLabs agent
- Bidirectional PCM on AgentStream
- A production-oriented Python bridge under `integrations/elevenlabs/exotel/bridge.py`

Before you start

Item	Notes
ElevenLabs	Agent ID + API key
Exotel	Account SID, API Key, API Token, ExoPhone
Python 3.10+	Virtualenv recommended
Public WSS	cloudflared or ngrok for the first test

Audio tip: In the ElevenLabs agent, set output to `pcm_8000` when available. That avoids a 16→8 kHz resample and reduces "slow" or muddy voice on the phone.

Install

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream

cp shared/env.exotel.example shared/.env.exotel
# Fill EXOTEL_ACCOUNT_SID, EXOTEL_API_KEY, EXOTEL_API_TOKEN, EXOTEL_CALLER_ID

cd integrations/elevenlabs
python3 -m venv venv && source venv/bin/activate
pip install -r exotel/requirements.txt

export ELEVENLABS_AGENT_ID="your_agent_id"
export ELEVENLABS_API_KEY="your_api_key"
export ELEVENLABS_REGION="default"   # default | us | eu | india
export BG_SOUND_VOLUME="0"           # 0 = no background ambience while testing
```

Run the bridge and open a tunnel

Terminal A:

```bash
cd integrations/elevenlabs
source venv/bin/activate
python exotel/bridge.py --port 10002 \
  --agent-id "$ELEVENLABS_AGENT_ID" \
  --api-key "$ELEVENLABS_API_KEY"
```

The bridge listens on `0.0.0.0:10002`. Path: `/v1/convai/conversation/exotel`.

Terminal B:

```bash
cloudflared tunnel --url http://127.0.0.1:10002
# Note the https://….trycloudflare.com host — use wss:// with the same host
```

Place a Connect call

From the **repo root**:

```bash
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_CLOUDFLARE_HOST/v1/convai/conversation/exotel"
```

StreamUrl	`wss://HOST/v1/convai/conversation/exotel`
Port	`10002`

Answer the handset. The agent should greet at normal pitch and respond when you speak.

Verify

- Voice is not slow or unusually deep (if it is, set agent output to `pcm_8000` or confirm resample logs).
- Bridge logs show media flowing and a `first_audio_ms=…` style line when audio starts.
- Call lasts well beyond a few seconds (very short hangups usually mean a bad StreamUrl or the receive loop blocked).

Go-live checklist

- [ ] Bridge runs on a stable host with a real TLS certificate for `wss://`
- [ ] API keys only in env / secret store
- [ ] Agent audio format set for telephony (`pcm_8000` preferred)

- [ ] Background sound volume set intentionally (or `0`)
- [ ] Monitoring on process health and WebSocket disconnects
- [ ] Load-tested concurrent calls for your expected traffic

This sample is a solid bridge for pilots. High concurrency still needs horizontal scale (multiple bridge instances behind a WSS load balancer), timeouts, and ops metrics.

Troubleshoot

Symptom	What to check
Call drops in ~4 seconds	StreamUrl path wrong; tunnel down; bot not accepting WSS
Slow / deep voice	Agent still on 16 kHz PCM played as 8 kHz – prefer `pcm_8000` or confirm bridge resample
Agent parrots the caller	Prompt / conversation override and half-duplex behaviour in the bridge README
No agent audio	API key, agent ID, region, and ElevenLabs console errors

Related

- Repo README: `integrations/elevenlabs/README.md`
- **Connect Voice AI API**
- **OpenAI Realtime-AgentStream Integration**
- **Sarvam AI- AgentStream Integration**

4.1.2. Sarvam AI- AgentStream Integration

Exotel AgentStream

Run an Indian-language voice bot on Exotel using **Sarvam Saaras** (STT) and **Bulbul** (TTS). This is a cascaded pipeline (speech → text → speech), not native speech-to-speech.

Sample code: `integrations/sarvam` in the **Agent-Stream** repo.

If you have not set up Connect yet, start with **Connect Voice AI with AgentStream**.

What you get

- AgentStream WebSocket server (`server.py`) that greets the caller and echoes turns via Sarvam
- Greeting audio **pre-synthesized at process start** so time-to-first-audio after `start` is usually under one second
- Paced PCM frames aligned with AgentStream expectations (~100 ms)

For a deeper production write-up, see `integrations/sarvam/SARVAM_AGENTSTREAM_INTEGRATION.md` in the repo.

Before you start

Item	Notes
Sarvam	`SARVAM_API_KEY` from the Sarvam dashboard
Exotel	Account SID, API Key, API Token, ExoPhone
Python 3.10+	Virtualenv recommended
Public WSS	cloudflared or ngrok for the first test

Install

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream

cp shared/env.exotel.example shared/.env.exotel
# Fill EXOTEL_* values

cd integrations/sarvam
python3 -m venv venv && source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env
```

Edit `.env` :

Bash

```
SARVAM_API_KEY=sk_...
GREETING_TEXT=Namaste! Please say something after the beep.
SAMPLE_RATE=8000
SERVER_PORT=8000
```

Run the server and open a tunnel

Terminal A (from `integrations/sarvam`):

Bash

```
source venv/bin/activate
export PYTHONPATH="$(cd ../.. && pwd)"
python server.py
```

You should see `Greeting cache ready…` and a listener on `0.0.0.0:8000`. WebSocket path: `/ws`.

Terminal B:

Bash

```
cloudflared tunnel --url http://127.0.0.1:8000
```

Place a Connect call

From the **repo root**:

Bash

```
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_CLOUDFLARE_HOST/ws?sample-rate=8000"
```

StreamUrl	`wss://HOST/ws?sample-rate=8000`
Port	`8000`

Answer the phone. The greeting should play quickly. Speak a short sentence; watch `user:` / `agent:` style logs and hear the reply.

Verify

- Boot: `Greeting cache ready`
- Call: `first_audio_ms` typically under 1000 for the cached greeting

- After you speak: STT text and TTS playback; clean `stop` when the call ends

Note: generative replies still wait on Sarvam HTTP TTS (~1–2 seconds). Only the greeting is pre-warmed.

Go-live checklist

- [ ] Server on a stable host with valid TLS for `wss://`
- [ ] `SARVAM_API_KEY` only in secrets
- [ ] StreamUrl includes `/ws?sample-rate=8000`
- [ ] Greeting text and language settings match your market
- [ ] Timeouts and concurrency limits sized for Sarvam API quotas
- [ ] Consider caching common prompts if you need lower reply latency

Troubleshoot

Symptom	What to check
No greeting	Boot cache failed – check API key and `GREETING_TEXT`; look for cache errors
Hangup in a few seconds	Wrong StreamUrl (must include `/ws`); tunnel down
Slow replies after you speak	Expected for HTTP STT/TTS; shorten prompts or cache TTS
Wrong language	Sarvam language / speaker settings in pipeline config

Related

- Repo README: `integrations/sarvam/README.md`
- Full guide: `integrations/sarvam/SARVAM_AGENTSTREAM_INTEGRATION.md`
- **Connect Voice AI API**
- **OpenAI Realtime-AgentStream Integration**
- **ElevenLabs's ElevenAgents -AgentStream Integration**

4.1.3. Build a modular voice agent with Pipecat

Build a modular voice agent with Pipecat on Exotel AgentStream

Wire Exotel phone audio to a **Pipecat** pipeline using the official `ExotelFrameSerializer`. Default stack: **Deepgram STT → OpenAI LLM → Cartesia TTS** at **8 kHz PCM**. Swap any stage in `bot.py` without rewriting the Exotel WebSocket glue.

Sample code: `integrations/pipecat` in the **Agent-Stream** repo.

If you have not set up Connect yet, start with **Connect Voice AI with AgentStream**.

When to use Pipecat (vs speech-to-speech)

	Pipecat cascade	OpenAI Realtime / ElevenLabs
Shape	STT → LLM → TTS (you own each hop)	Native speech-to-speech
Best for	Swappable vendors, custom tools, cost control per stage	Lowest conversational latency and barge-in feel
Expect	~turn detection + LLM + TTS before reply audio	Model speaks as it “thinks”

Use this guide when you want a **composable** AgentStream bot. For the most natural phone talk, prefer the **OpenAI Realtime** or **ElevenLabs** guides.

What you get

- FastAPI WebSocket host that speaks Exotel AgentStream (`/ws`)
- Pipecat `ExotelFrameSerializer` (16-bit linear PCM, typically 8 kHz – not μ-law)
- Boot-time **Cartesia greeting cache** (same voice ID as live TTS) for near-instant first media
- Tuned turn-taking: Silero VAD + Local Smart Turn (sub-second silence stop)
- Official Pipecat Exotel notes: **Exotel WebSockets**

Before you start

Item	Notes
OpenAI	`OPENAI_API_KEY` (LLM replies)
Deepgram	`DEEPGRAM_API_KEY` (STT)
Cartesia	`CARTESIA_API_KEY` (+ optional `CARTESIA_VOICE_ID`)
Exotel	Account SID, API Key, API Token, ExoPhone
Python 3.10+	Virtualenv recommended
Public WSS	cloudflared or ngrok for the first test

Install

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream

cp shared/env.exotel.example shared/.env.exotel
# Fill EXOTEL_* values

cd integrations/pipecat
python3 -m venv venv && source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env
```

Edit `.env` (never commit this file):

Bash

```
OPENAI_API_KEY=sk-...
OPENAI_MODEL=gpt-4o-mini
DEEPGRAM_API_KEY=...
CARTESIA_API_KEY=sk_car_...
CARTESIA_VOICE_ID=71a7ad14-091c-4e8e-a314-022ece01c121
CARTESIA_MODEL=sonic-3.5
PIPECAT_GREETING=Hi, how can I help?
SMART_TURN_STOP_SECS=0.8
SERVER_HOST=0.0.0.0
SERVER_PORT=8765
```

Keep `CARTESIA_VOICE_ID` / `CARTESIA_MODEL` / 8 kHz PCM the same for the greeting cache and live TTS so the agent voice stays consistent.

Run the bot and open a tunnel

Terminal A:

Bash

```
cd integrations/pipecat
source venv/bin/activate
python server.py
```

On startup you should see:

- `Cartesia TTS websocket warmed`
- `Greeting cache ready …`
- `Pipecat Exotel bridge ready — WSS /ws`
- Uvicorn on `0.0.0.0:8765`

WebSocket path: `/ws`.

Terminal B:

```bash
cloudflared tunnel --url http://127.0.0.1:8765
```

Place a Connect call

From the **repo root**:

```bash
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_CLOUDFLARE_HOST/ws?sample-rate=8000"
```

StreamUrl	`wss://HOST/ws?sample-rate=8000`
Port	`8765`

The `?sample-rate=8000` query parameter is required for PSTN. Answer the phone: you should hear the cached greeting quickly, then speak a short turn and get an LLM + Cartesia reply.

Architecture (what runs on each call)

```text
Caller ⟷ Exotel AgentStream (8 kHz PCM)
              ↕ wss://…/ws
         FastAPI + ExotelFrameSerializer
              ↕
         Pipecat pipeline
           Deepgram STT → user aggregator (VAD + Smart Turn)
             → OpenAI LLM → Cartesia TTS → Exotel media out
```

Piece	Role
`server.py`	Accept Exotel WSS, build transport + serializer, run one bot session
`bot.py`	Pipeline, VAD / Smart Turn, LLM, Cartesia TTS
`greeting_cache.py`	Boot synthesize + silence trim; push PCM on connect
`voice_config.py`	Shared voice / rate / encoding for cache and live TTS

Pipecat does **not** auto hang-up via Exotel REST in this sample – ending the WebSocket ends the media bridge. See the **serializer docs**.

Verify

- Boot: `Greeting cache ready` and `voice=<same CARTESIA_VOICE_ID>`
- Connect: `cached greeting … queue_ms=0`
- After you speak: Smart Turn completes in under ~1s of silence (not ~3s)
- Reply audio sounds natural (sentence-level Cartesia), not stretched word-by-word
- Pitch matches a normal phone call (8 kHz end-to-end, no μ-law mix-up)

Latency expectations (honest)

Stage	Typical	Notes
First greeting	Near 0 ms after pipeline ready	Pre-cached Cartesia PCM
End-of-user-turn	~0.8s silence (configurable)	`SMART_TURN_STOP_SECS` (Pipecat default is 3.0 – too slow for phones)
LLM TTFB	~0.5–2s	Use `gpt-4o-mini` (or similar) for voice
Cartesia TTFA	~0.2s	Plus small leading silence from the model

This is still a **cascade**. Speech-to-speech providers will feel snappier for free-form chat.

Go-live checklist

- [ ] Bot on a stable host with valid TLS for `wss://`
- [ ] API keys only in secrets – not in git
- [ ] StreamUrl is `wss://HOST/ws?sample-rate=8000`
- [ ] Greeting text / voice ID match your brand
- [ ] `SMART_TURN_STOP_SECS` tuned on real handsets (too low cuts users off mid-thought)
- [ ] Cartesia uses **sentence** aggregation for natural speech (do not force token flush with `max_buffer_delay_ms=0` unless you accept choppy audio)
- [ ] Process supervision and clean disconnect handling
- [ ] Capacity test: one process ≠ unlimited concurrent calls

Troubleshoot

Symptom	What to check
No greeting	Boot cache failed – `CARTESIA_API_KEY`, network; look for `Greeting warm failed`
Silence after connect	Tunnel URL, path `/ws`, `?sample-rate=8000`
~3s dead air before every reply	Smart Turn still at default 3s – set `SMART_TURN_STOP_SECS=0.8`
“Hello……how……can……” stretched speech	Token-level TTS flush – use sentence aggregation (sample default)
Deep / slow voice	Sample-rate mismatch – keep pipeline and Cartesia at 8000 PCM
Hangup in a few seconds	Wrong StreamUrl or tunnel died
Slow LLM	Switch `OPENAI_MODEL` to a low-latency chat model; shorten system prompt

Swap STT / LLM / TTS

Edit `integrations/pipecat/bot.py`. Keep:

- `audio_in_sample_rate` / `audio_out_sample_rate` = `8000`
- Transport + `ExotelFrameSerializer` unchanged in `server.py`
- Greeting cache voice settings in sync if you change Cartesia (or re-point cache at your new TTS)

Upstream patterns: **pipecat-examples/exotel-chatbot**.

Related

- Repo README: `integrations/pipecat/README.md`
- **Pipecat ExotelFrameSerializer**
- **Connect Voice AI API**
- **OpenAI Realtime** · **ElevenLabs** · **Sarvam**

4.1.4. Cartesia Agents with Exotel AgentStream

Connect an Exotel phone call to a deployed **Cartesia** voice agent over AgentStream. Cartesia runs the full speech-to-speech stack (STT, LLM, TTS, turn-taking). Your bridge only moves PCM between Exotel and Cartesia's **Agents WebSocket API**.

Sample code: `integrations/agents/cartesia-line-native` in the **Agent-Stream** repo.

If you have not set up Connect yet, start with **Connect Voice AI with AgentStream**.

What you get

- Outbound (or inbound via applet) calls that talk to your Cartesia Line agent (`agent_…`)
- Bidirectional AgentStream PCM at 8 kHz (telephony default)
- Uplink gate so early line noise does not barge into the greeting
- Access-token cache + startup prewarm to cut first-media latency

```text
Caller ⟷ Exotel ←WSS PCM→ FastAPI /ws ←WSS→ Cartesia Line agent
```

This is **not** the same as using Cartesia Sonic as TTS inside a cascaded STT → LLM → TTS pipeline (for example Pipecat or `deepgram-gemini-cartesia-native`). Here the Line agent owns the conversation.

Before you start

Item	Notes
Cartesia	API key (`sk_car_…`) + deployed Line agent ID (`agent_…`)
Exotel	Account SID, API Key, API Token, ExoPhone
Python 3.10–3.12	Prefer 3.12 (`audioop` + `websockets` work cleanly)
Public WSS	cloudflared or ngrok for the first test

Audio tip: Default wire format is Cartesia `mulaw_8000` (μ-law at 8 kHz). Exotel AgentStream stays PCM16 @ 8 kHz; the bridge converts both ways with no resample. That keeps pitch/speed correct on the handset.

Install

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream

cp shared/env.exotel.example shared/.env.exotel
# Fill EXOTEL_ACCOUNT_SID, EXOTEL_API_KEY, EXOTEL_API_TOKEN, EXOTEL_CALLER_ID

cd integrations/agents/cartesia-line-native
python3.12 -m venv venv && source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env
```

Edit `.env` (never commit this file):

Bash

```
CARTESIA_API_KEY=sk_car_...
CARTESIA_AGENT_ID=agent_...
SERVER_PORT=4055
# Optional: CARTESIA_UPLINK_GATE_MS=2500
```

Smoke-test Cartesia alone (no phone):

Bash

```
python test_ws_connection.py
```

You should see `ack`, then `media_output` / transcript deltas from the agent greeting.

Run the bridge and open a tunnel

Terminal A:

Bash

```
cd integrations/agents/cartesia-line-native
source venv/bin/activate
python server.py
```

On startup you should see `Cartesia access token prewarmed`, then Uvicorn on `0.0.0.0:4055`. WebSocket path: `/ws`.

Terminal B:

Bash

```
cloudflared tunnel --url http://127.0.0.1:4055
# or: ngrok http 4055
```

Place a Connect call

From the **repo root**:

Bash

```
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_HOST/ws?sample-rate=8000"
```

StreamUrl	`wss://HOST/ws?sample-rate=8000`
Port	`4055`
StreamType	`bidirectional`

Answer the handset. The Line agent should greet within about **1–1.5s** of AgentStream `start` (after token prewarm), then respond when you speak.

Auth (Cartesia)

Server-side flow matches Cartesia docs:

1. `POST https://api.cartesia.ai/access-token` with `grants: { "agent": true }`
2. Connect `wss://api.cartesia.ai/agents/stream/{agent_id}` with `Authorization: Bearer <token>` and `Cartesia-Version: 2025-04-16`
3. First message must be `start` (`input_format` , optional `voice_id` / agent overrides)
4. Stream `media_input` / receive `media_output` , handle `clear` , conversation events

This sample caches the token and refreshes it before expiry so later calls skip the HTTP RTT.

Latency notes

Without gating, first audio can land around **4s** because:

1. Cold access-token + WebSocket connect (~1s)
2. Early Exotel uplink noise is treated as a user turn → Cartesia `clear` delays the greeting (~2–3s)

The sample mitigates both (token prewarm + uplink gate). Useful log lines:

- `ready_ms=…` / `token_ms=…` / `ws_ms=…` / `ack_ms=…`
- `cartesia_first_media_out_ms=…` / `since_ack_ms=…`
- `first_audio_ms=…`
- `uplink_open reason=assistant_turn_ended|gate_timeout`

Tune `CARTESIA_UPLINK_GATE_MS` (default `2500`). Set `0` to disable the gate.

Verify

- `test_ws_connection.py` prints `ack` + greeting audio/text
- Connect call logs show `Cartesia ready … call_id=ac_…` then `turn_started … role=assistant`
- Voice is normal pitch/speed (not slow/deep)
- Call lasts well beyond a few seconds

Go-live checklist

- [ ] Bridge on a stable host with real TLS for `wss://`
- [ ] API keys only in env / secret store
- [ ] `CARTESIA_AGENT_ID` points at the production Line deployment
- [ ] Uplink gate tuned for your greeting length
- [ ] Monitoring on process health, Cartesia disconnect reasons, `first_audio_ms`
- [ ] Load-tested concurrent calls for expected traffic

Troubleshoot

Symptom	What to check
Call drops in ~4 seconds	StreamUrl path / `?sample-rate=8000`; tunnel; bot accepting WSS
Slow / deep voice	Confirm Exotel is 8 kHz and Cartesia format is `mulaw_8000` (or matching PCM rate)
Greeting delayed / cut off	Uplink gate (`CARTESIA_UPLINK_GATE_MS`); look for false `turn_started role=user` before assistant
No agent audio	API key, agent ID, `access-token` with `agent` grant, Cartesia console / call record
`ack` timeout	Agent deployment status; `Cartesia-Version` header; network to `api.cartesia.ai`

Related

- Recipe README: `integrations/agents/cartesia-line-native/README.md`
- **Cartesia Agents WebSocket API**
- **Connect Voice AI API**

4.1.5. OpenAI Realtime-AgentStream Integration

Connect an Exotel phone call to **OpenAI Realtime** (speech-to-speech). Exotel stays at **8 kHz PCM**. The bridge talks to OpenAI as **PCM 16-bit at 24 kHz** and resamples both directions locally.

Sample code: `integrations/openai-realtime` in the **Agent-Stream** repo.

If you have not set up Connect yet, start with **Connect Voice AI with AgentStream**.

What you get

- True speech-to-speech (no separate STT → LLM → TTS pipeline in your code)
- Instant pre-cached greeting on call start (optional, on by default)
- Paced AgentStream media frames suitable for Connect Voice AI

OpenAI does **not** accept linear PCM at 8 kHz. Do not set `audio/pcm` with `rate: 8000`. Use 24 kHz PCM on the OpenAI wire (this sample) or G.711 μ-law at 8 kHz if you deliberately implement that path.

Before you start

Item	Notes
OpenAI	API key with Realtime access
Exotel	Account SID, API Key, API Token, ExoPhone
Python 3.10+	On 3.13+, `audioop-lts` is pulled in via requirements
Public WSS	cloudflared or ngrok for the first test

Install

Bash

```
git clone https://github.com/exotel/Agent-Stream.git
cd Agent-Stream

cp shared/env.exotel.example shared/.env.exotel
# Fill EXOTEL_* values

cd integrations/openai-realtime
python3 -m venv venv && source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env
```

Edit `.env` (never commit this file):

Bash

```
OPENAI_API_KEY=sk-...
OPENAI_MODEL=gpt-realtime
OPENAI_VOICE=coral
SAMPLE_RATE=8000
DEFAULT_SAMPLE_RATE=8000
SERVER_PORT=5000
COMPANY_NAME=Your Company
SALES_BOT_NAME=Sara
SEND_TEST_TONE=false
INSTANT_GREETING=true
```

Run the bot and open a tunnel

Terminal A:

Bash

```
cd integrations/openai-realtime
source venv/bin/activate
python main.py
```

On startup you should see a greeting cache message, then `server listening on 0.0.0.0:5000`. WebSocket path is `/` (root).

Terminal B:

Bash

```
cloudflared tunnel --url http://127.0.0.1:5000
```

Place a Connect call

From the **repo root**:

Bash

```
set -a && source shared/.env.exotel && set +a

python shared/place_connect_call.py \
  --to +91XXXXXXXXXX \
  --stream-url "wss://YOUR_CLOUDFLARE_HOST/?sample-rate=8000"
```

StreamUrl	`wss://HOST/?sample-rate=8000`
Port	`5000`

The `?sample-rate=8000` query parameter is required for PSTN. Answer the phone: you should hear the greeting quickly, then be able to talk turn-by-turn.

Verify

- Logs: `INSTANT greeting` (or Realtime greeting fallback), `Audio Format: audio/pcm → audio/pcm`, `first_audio_ms=…`
- Pitch sounds like a normal phone call (not stretched or deep)
- After you speak, the model replies; barge-in cancels playback when configured

Go-live checklist

- [ ] Bot on a stable host with valid TLS for `wss://`
- [ ] `OPENAI_API_KEY` only in secrets – not in git
- [ ] `SEND_TEST_TONE=false` in production
- [ ] StreamUrl always includes `?sample-rate=8000` for 8 kHz telephony
- [ ] Voice and instructions tuned for short phone turns
- [ ] Process supervision and disconnect handling

This tree is a working sample for pilots. Large concurrent load needs multiple processes, a WSS edge, and capacity testing – it is not a multi-tenant autoscaler by itself.

Troubleshoot

Symptom	What to check
Deep / slow voice	Old μ-law/PCM mismatch – current sample uses PCM 24 kHz ↔ 8 kHz; pull latest `core/bot.py`
Beep before greeting	`SEND_TEST_TONE=true` – set to `false`
No audio	Tunnel URL, `?sample-rate=8000`, OpenAI key / Realtime access
Long delay to first speech	Instant greeting cache failed (check boot logs); otherwise Realtime TTFA is often ~1–3s after connect without cache

Related

- Repo README: `integrations/openai-realtime/README.md`
- **Connect Voice AI API**
- **ElevenLabs's ElevenAgents -AgentStream Integration**
- **Sarvam AI- AgentStream Integration**

4.2. OmniDimension Voicebot AgentStream Integration

1. Introduction

This document will help you walk through integrating Exotel phone numbers with OmniDimension (OmniDim) Voicebot, using call flows and importing into OmniDim.

What you'll achieve: incoming PSTN calls to Exotel → routed via configured call flow → handled by OmniDim VoiceBot.

2. Architecture Overview

```
Call flow:
PSTN call → Exotel → Custom call-flow (App Builder) → WebSocket / passthru endpoints → OmniDim agent (voicebot)
```

Sequence of events with endpoints:

1. Exotel receives the call on your Exophone number
2. Exotel triggers your configured "call flow" (App)
3. Within the flow: Voicebot applet, passthru applet(s), connect/hangup logic
4. Webhook / media endpoints point to OmniDim URLs
5. OmniDim handles the voice session, agent conversation, transfer or hangup

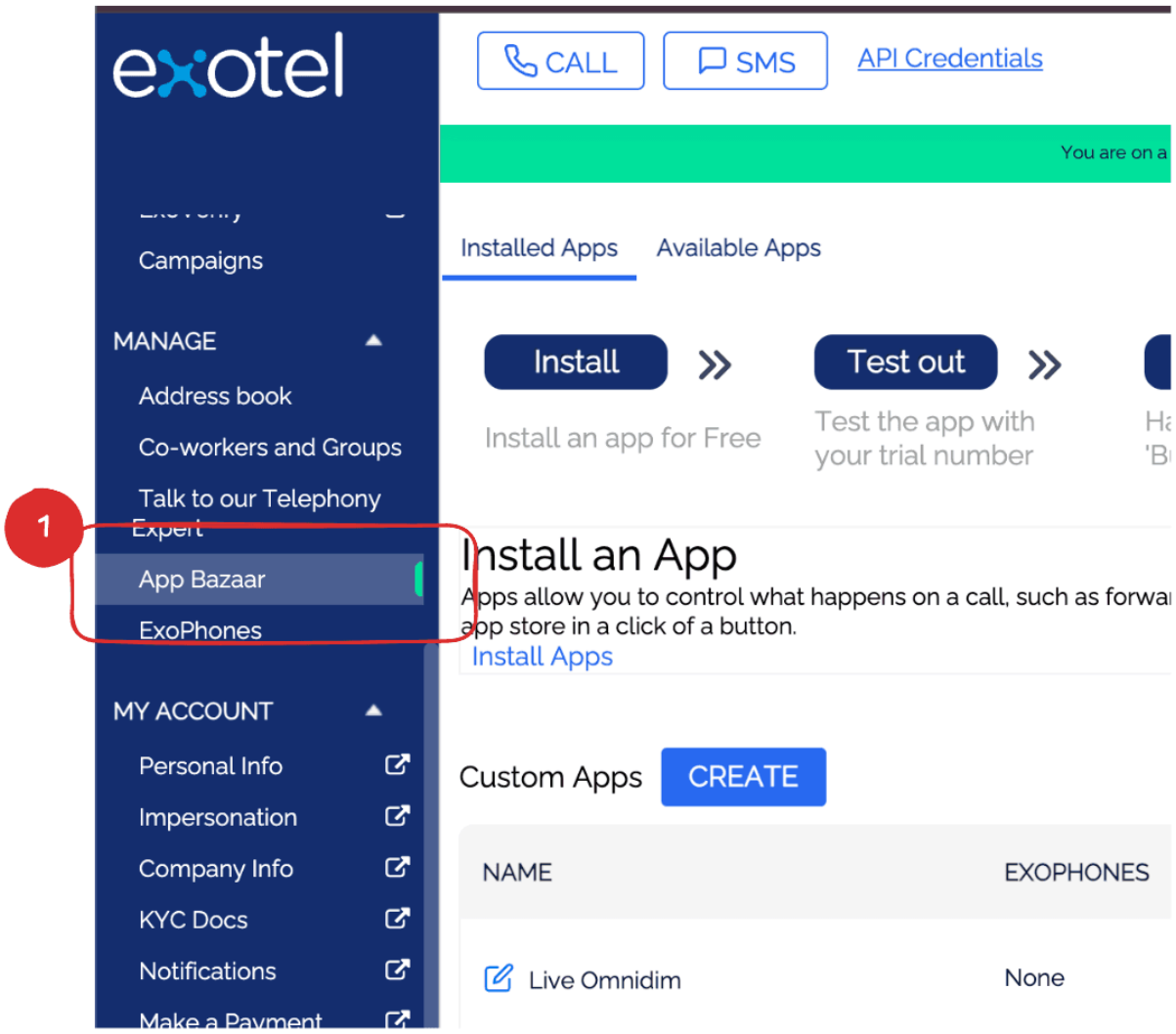
3. Prerequisites & Requirements

- Exotel account with KYC completed: **Steps to get started with Exotel**
- **An Exophone (140 series number/Landline numbers/Mobile DIDs) purchased & active on Exotel**
- **API Creds**
- Required credentials from Exotel:
  - API Key
  - API Token
  - Subdomain
  - Account SID
  - Exophone Number
  - App ID (call flow / App ID)
- Access to OmniDim portal & permission to import numbers
  URLs / endpoints in OmniDim for media/control.

4.Exotel Flow Setup

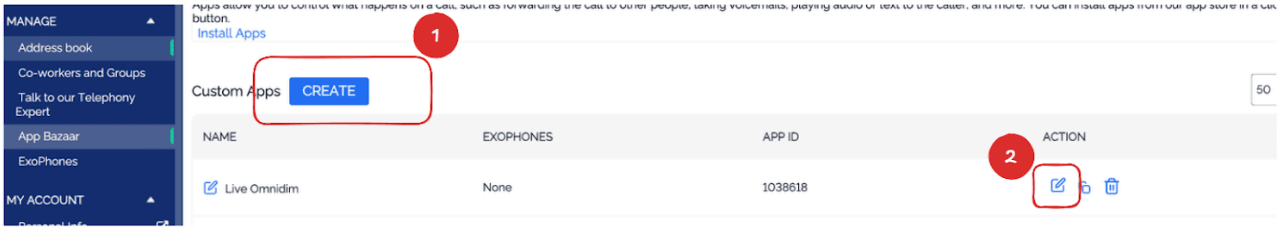
4.1 Access App Builder in Exotel

- In your Exotel dashboard, go to the side menu and click on 'App Bazaar'.
- Navigate to App Bazaar in the side menu



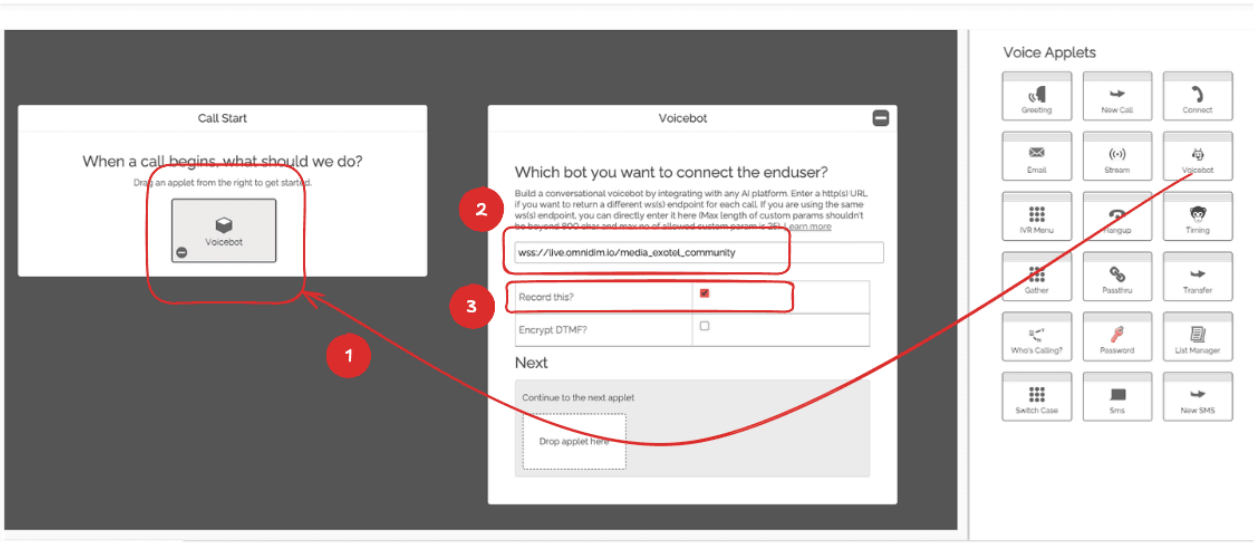
4.2 Create New App

- Click on 'Create New App' to start building your call flow.
- Click Create New App button



4.3 Add Voicebot Applet for Call Coming

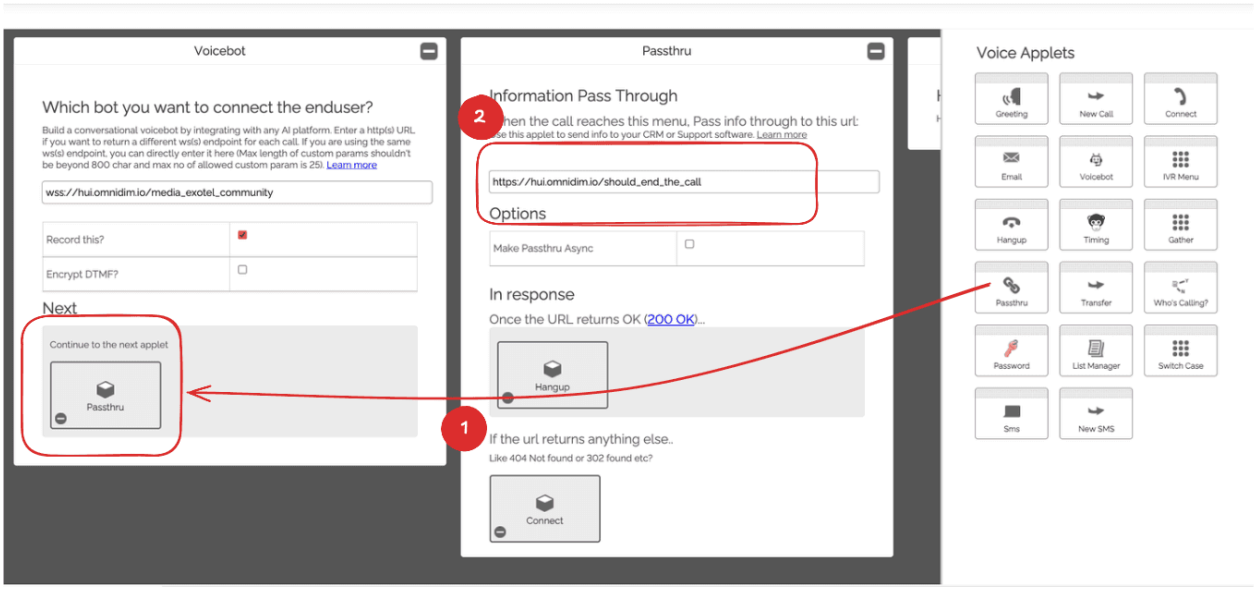
- On the 'Call Start' block, drag a 'Voicebot' applet from the right sidebar. In the URL field, enter: wss://live.omnidim.io/media_exotel_community and also turn recording ON.
- Drag the Voicebot applet and add the WSS URL



4.4 Add Second Passthru Applet

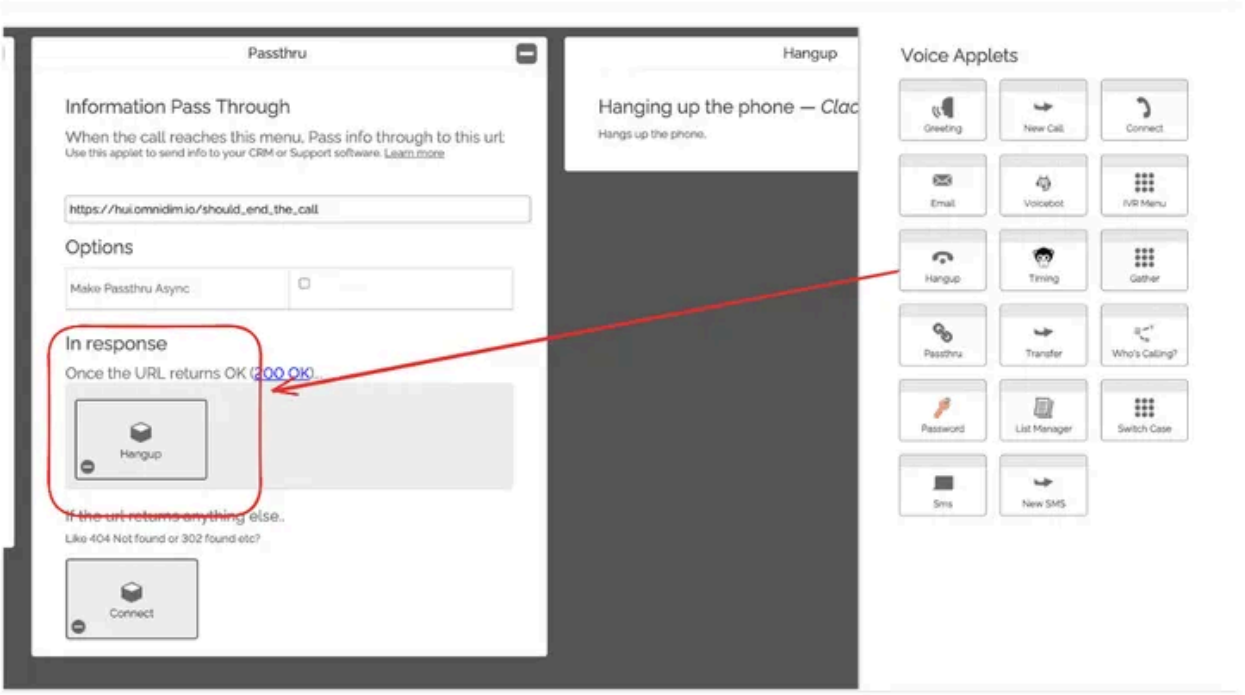
- On the voicebot applet's next section, drag another 'Passthru' applet. In the URL field, enter: https://live.omnidim.io/should_end_the_call

- Add second passthru applet



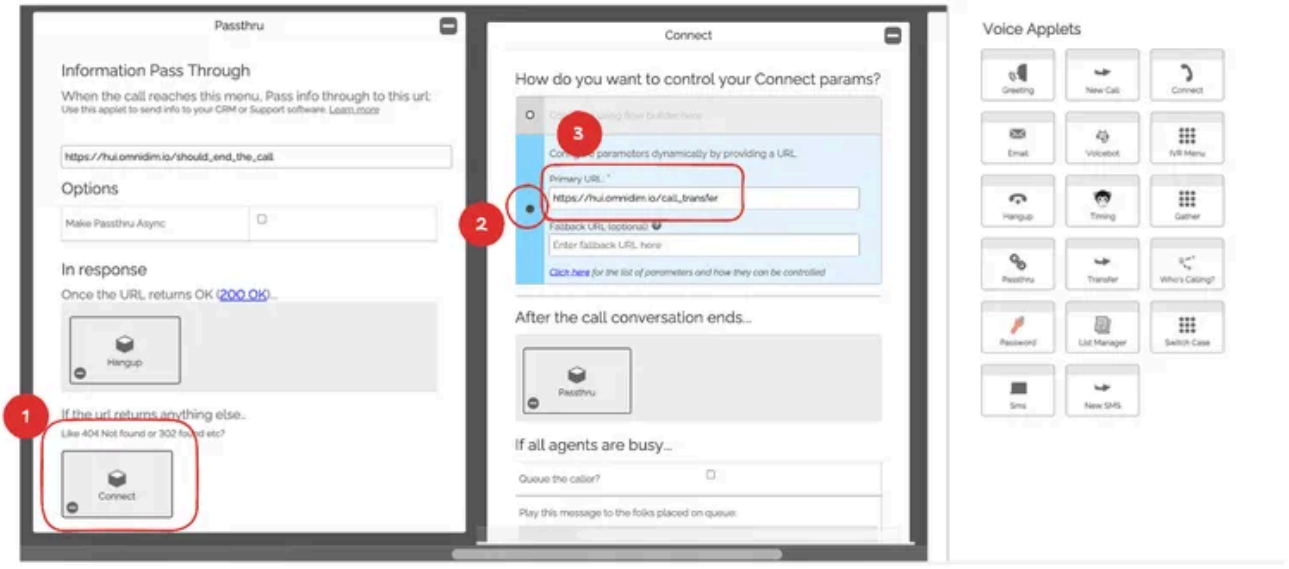
4.5 Add Hangup on 200 Response

- In the 200 response of the second passthru applet, drag a 'Hangup' applet to end the call when needed.
- Add hangup applet on 200 response



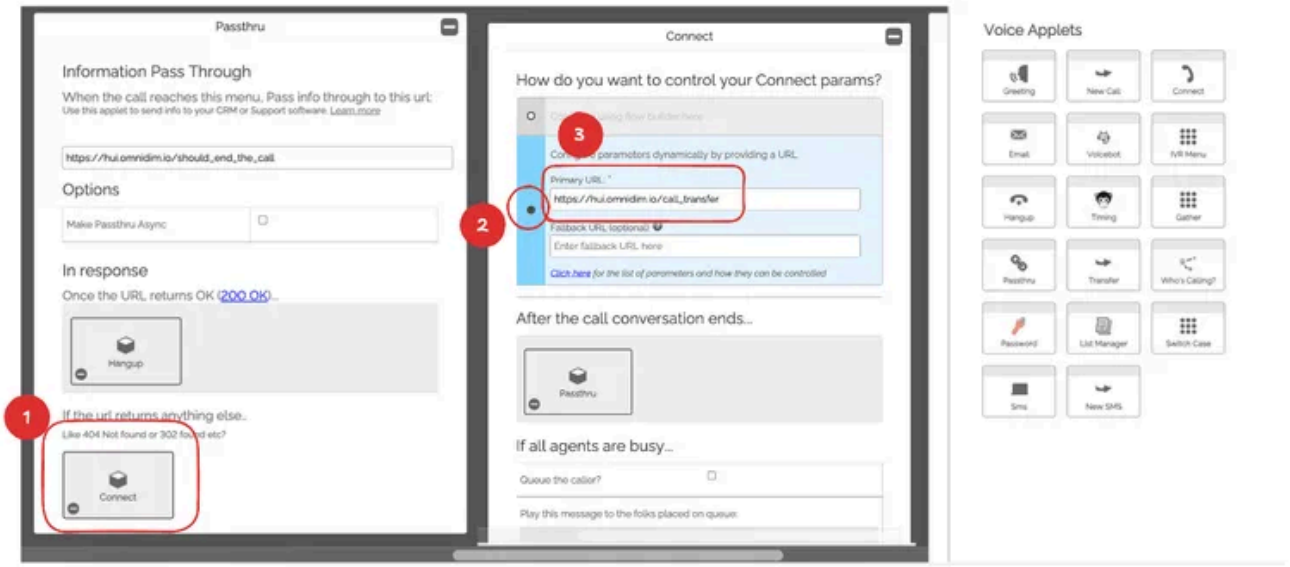
4.6 Add Connect Applet for Non 200 response

- In the 404 or 302 response of the second passthru applet, drag a 'Connect' applet for call transfers.
- Add connect applet on 404 or 302 response



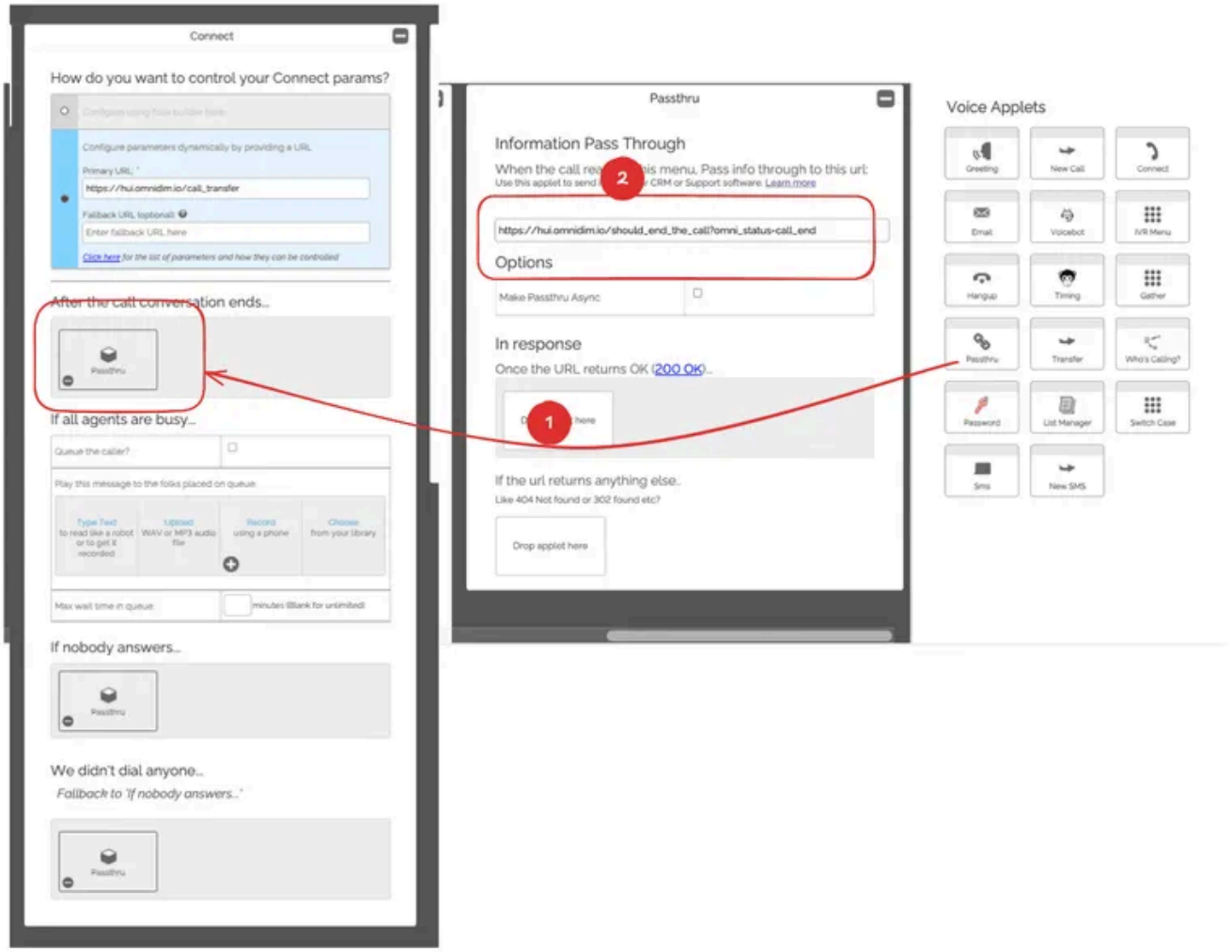
4.7 Configure Connect Applet

- In the connect applet, select 'Primary URL' and enter: https://live.omnidim.io/call_transfer
- Configure the Connect applet with the URL



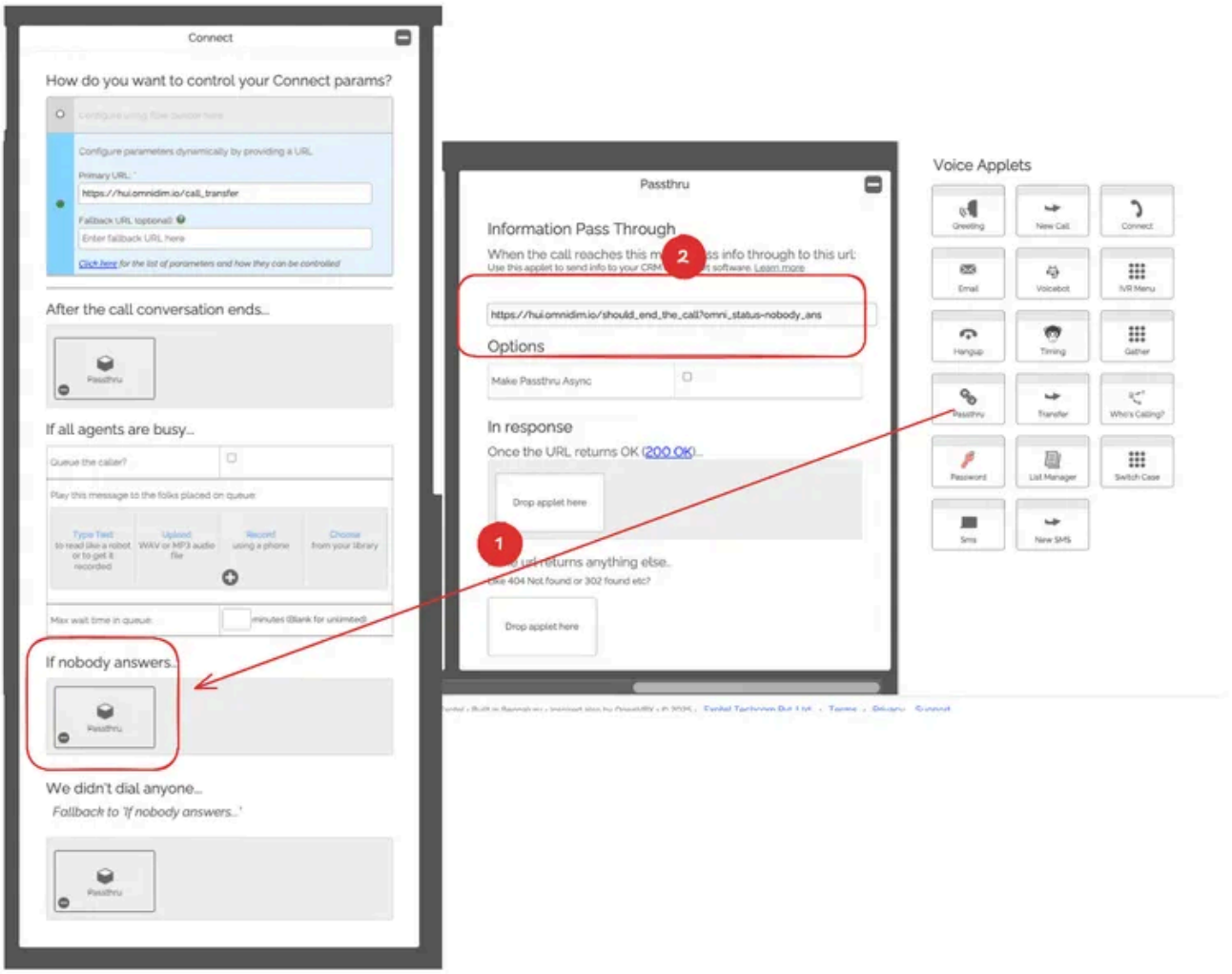
4.8 Add Passthru for Call End

- For the 'After the call conversation ends' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=call_end
- Add passthru for call end scenario



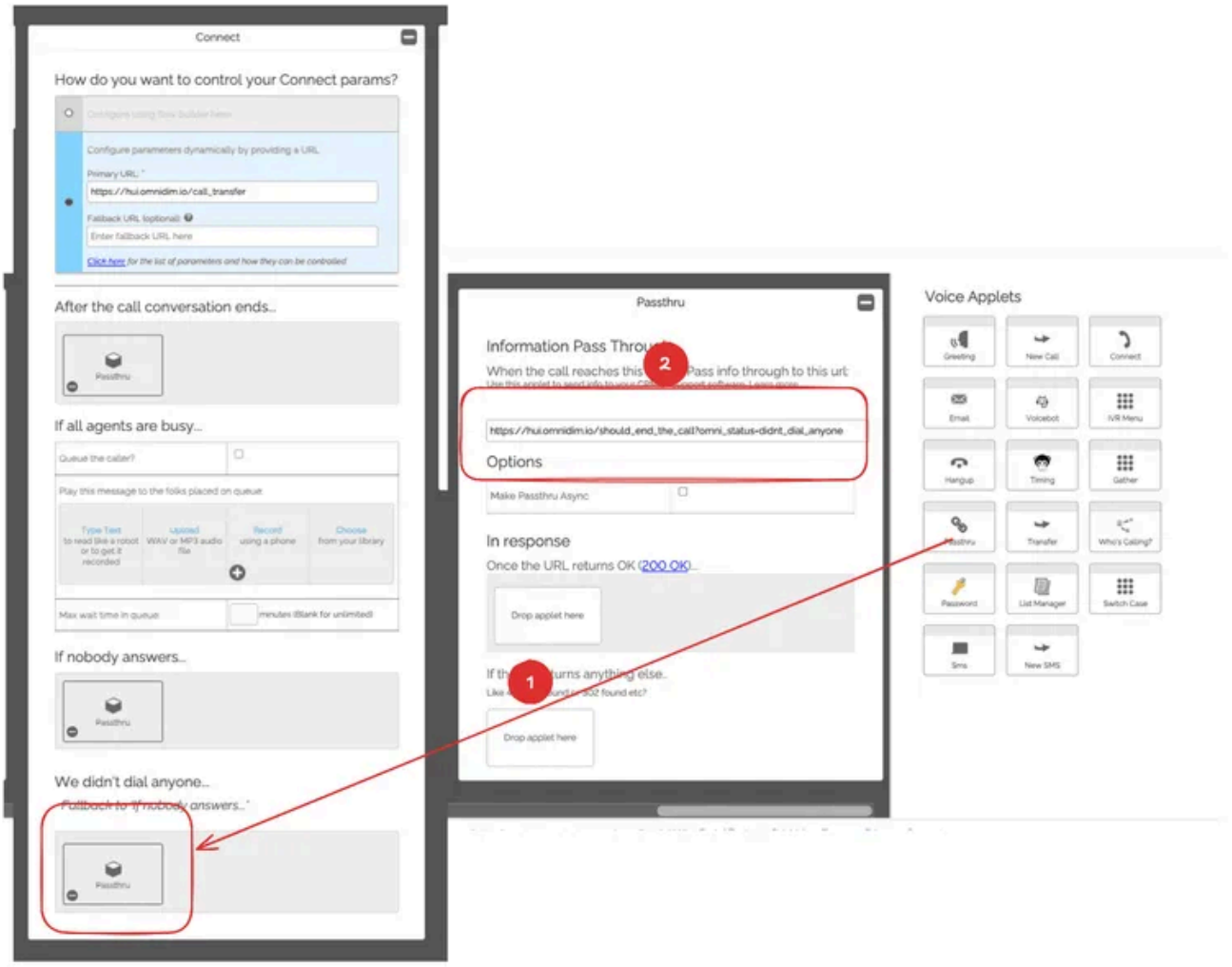
4.9 Add Passthru for No Answer

- For the 'If nobody answers' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=nobody_ans
- Add passthru for no answer scenario



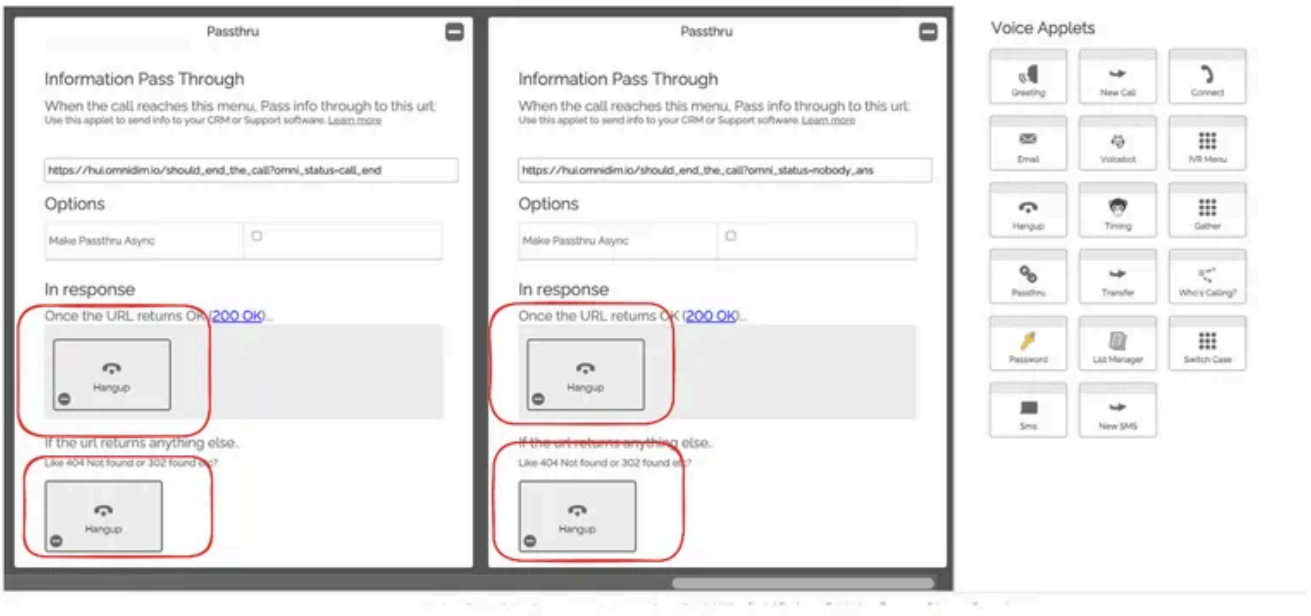
4.10 Add Passthru for No Dial

- For the 'We didn't dial anyone' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=didnt_dial_anyone
- Add passthru for no dial scenario



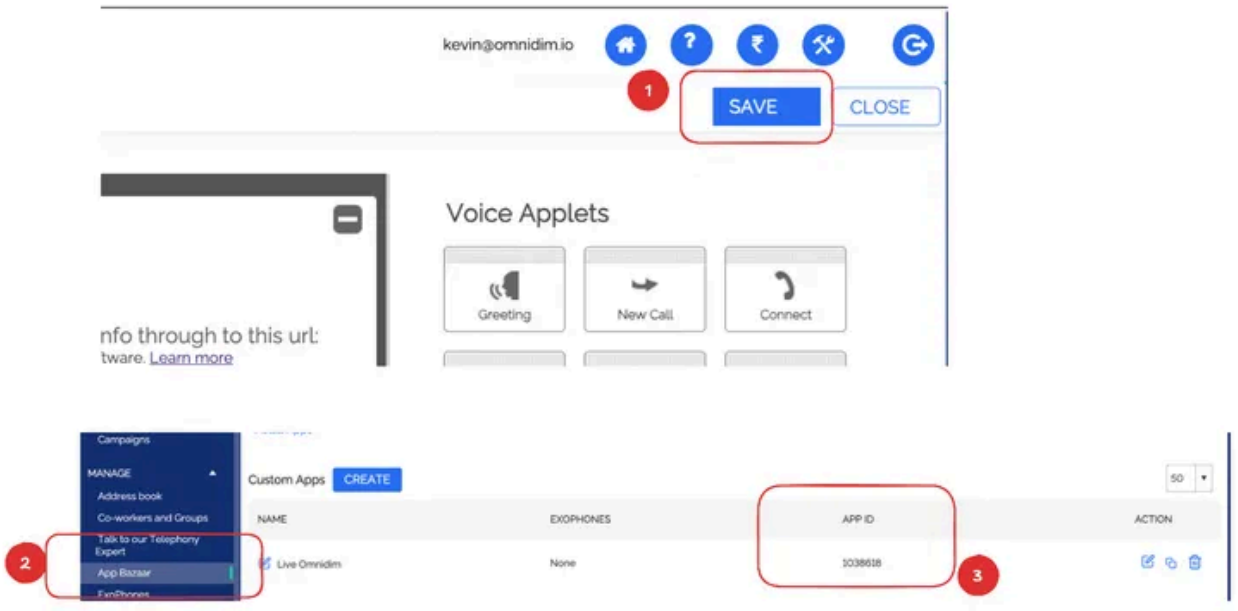
4.11 Add Hangup Applets

- On all the passthru applets you just created, add a 'Hangup' applet in both 200 and 302 responses to ensure calls end properly.
- Add hangup applets to all passthru responses



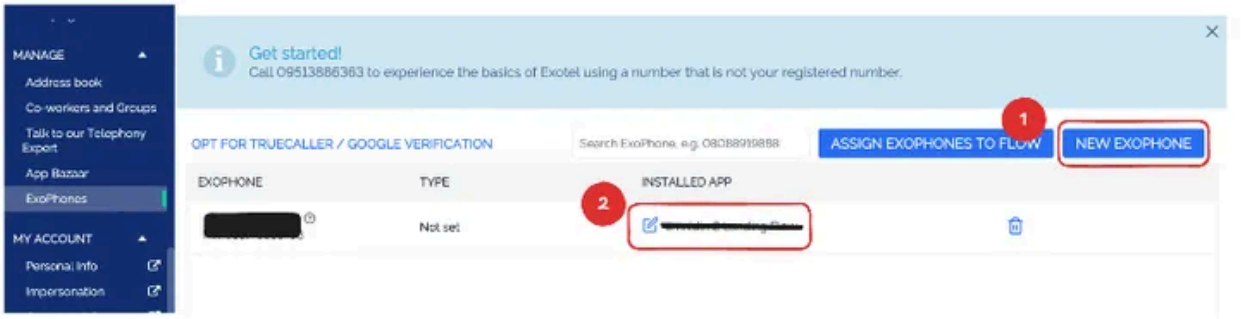
4.12 Save Your Call Flow

- Click 'Save' to save your call flow configuration and close the flow builder. Make sure to note down the App ID (call flow ID) as you'll need it for the import process.
- Save your call flow configuration



4.13 Attach Call Flow with your Exophone number

- Go to the Exophone section in the Exotel dashboard, buy a number and attach the call flow you just created. Make sure to note down the Phone Number as you'll need it for the import process.
- Attach the call flow to your Exophone number.



5. Importing the Number to OmniDim

- Go to Phone Numbers Section
- Log in to OmniDim dashboard → **Phone Numbers**
  Click “Import Exotel Phone Number”
- Opens an import form
  Fill in All Credentials
  - API Key
  - API Token
  - Subdomain
  - Account SID
  - Exophone number
  - App ID
- Submit Import
- Click **Import**
  On success, the phone number will be listed in OmniDim interface

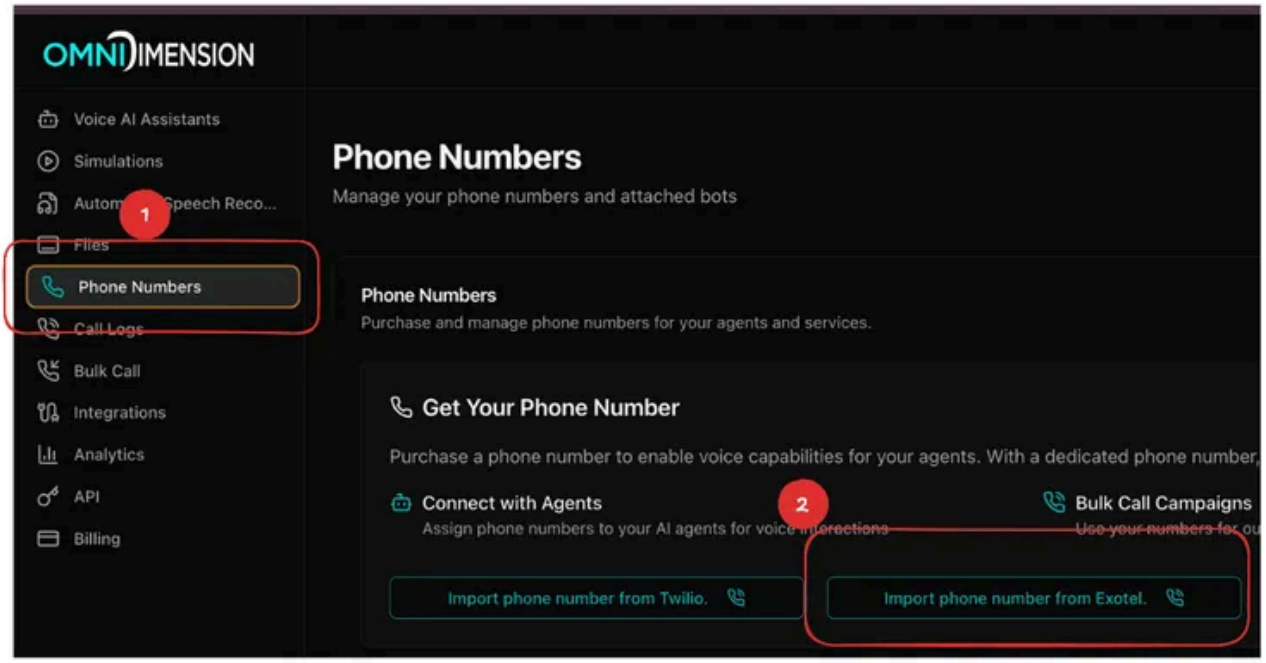
6. Import to OmniDimension

- Now that your call flow is set up in Exotel, follow these steps to import your phone number into OmniDimension:

6.1 Navigate to Phone Numbers

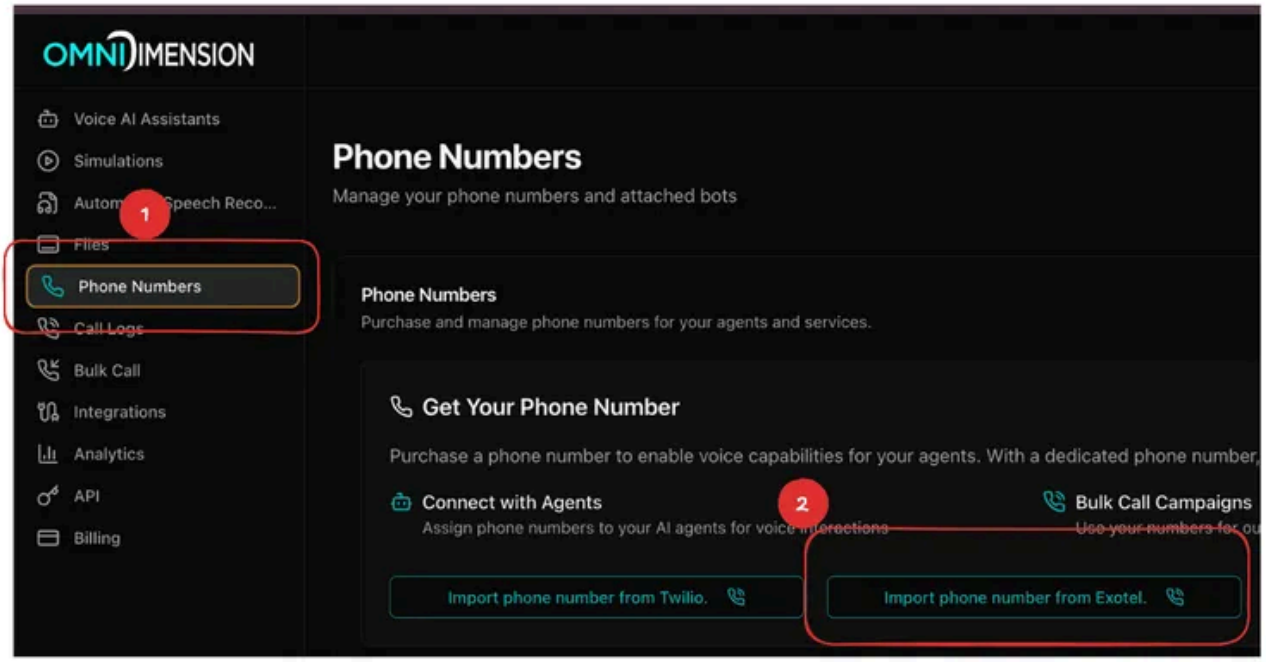
- In your OmniDimension dashboard, go to the 'Phone Numbers' section from the main navigation menu.

- Navigate to Phone Numbers in your dashboard



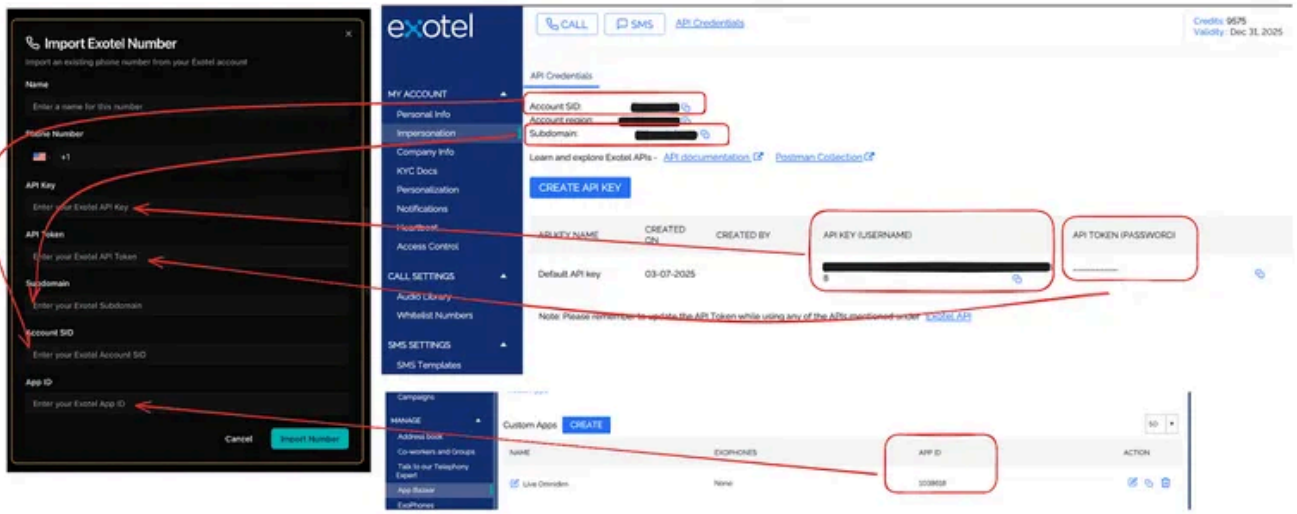
6.2 Click Import Exotel Phone Number

- Look for the 'Import Exotel Phone Number' button and click on it to open the import form.
- Click the Import Exotel Phone Number button



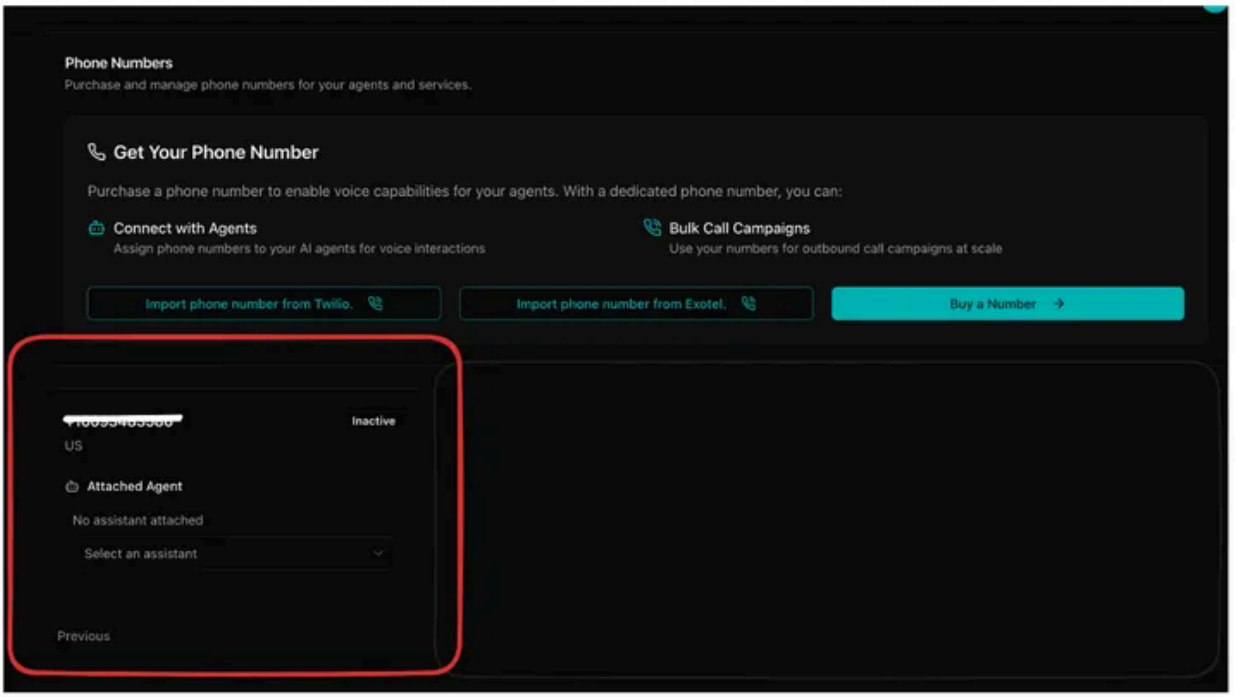
6.3 Fill in the Import Form: Enter all the credentials you collected from your Exotel dashboard in the form fields:

- Exotel API Key
- Exotel API Token
- Exotel Subdomain
- Exotel Account SID
- Exotel Phone Number
- Exotel App ID (your call flow ID)
- Fill in all the required credentials



6.4 Complete the Import

- Click the 'Import' button to complete the process. Your Exotel phone number will now appear in your OmniDimension dashboard and be ready for use with your AI agents.
- Your phone number has now been imported successfully

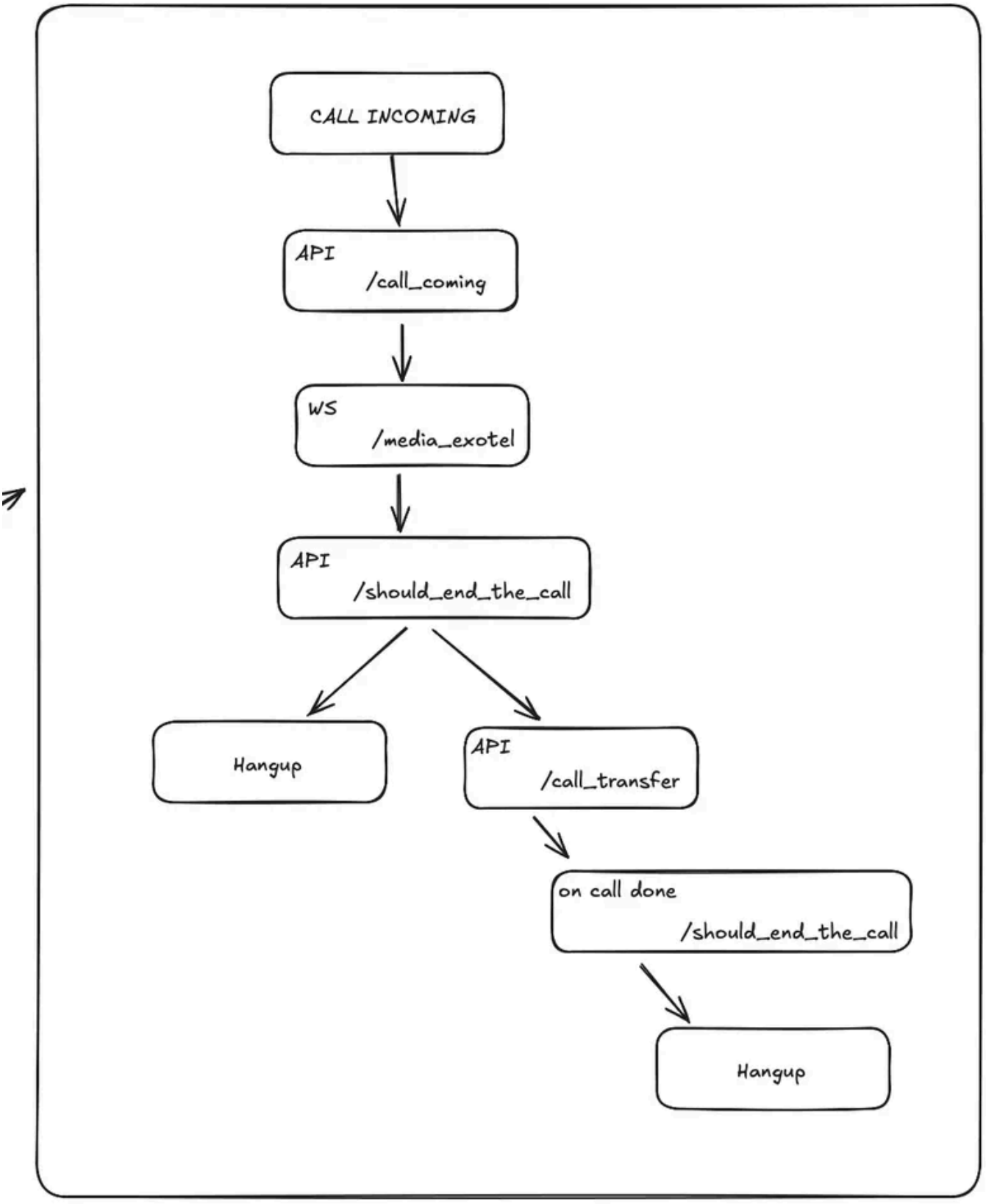


7. Understanding the Call Flow

Here's how your call flow works after setup. When someone calls your Exotel number:

- The call first goes to OmniDimension's call_coming endpoint
- OmniDimension then connects to your AI agent through the voicebot
- Your AI agent handles the conversation
- When the call should end, it goes to the should_end_the_call endpoint
- If a transfer is needed, it goes to the call_transfer endpoint
- All calls eventually end with a hangup action

Complete call flow overview



8. What Happens After Import

Once your phone number is successfully imported, you can:

- Use the phone number with any of your AI agents
- Receive incoming calls that are automatically handled by your agents
- Monitor call logs and performance in your dashboard

9. Next Steps

Now that your Exotel phone number is imported, here are some recommended next steps:

- Configure your AI agents to handle calls effectively
- Attach exotel phone number to your agent
- Test your setup by making a test call to your number

10. Troubleshooting Common Issues

If you encounter any issues during the setup process, here are some common solutions:

- Voicebot applet not visible: Make sure you've completed KYC verification. If still not visible, contact Exotel support
- Import fails: Double-check all credentials are correct and your call flow is properly configured
- Calls not connecting: Verify your call flow URLs are correct and your OmniDimension account is active
- Call hangup on connect: Make sure all URLs are entered exactly as shown with no extra spaces
- KYC issues: Contact Exotel support if you're having trouble with the verification process

11. Getting Help

If you need additional help with the Exotel integration:

- Check the Exotel documentation for detailed call flow setup
- Contact Exotel support for issues with your Exotel account or call flows
- Contact OmniDimension support for issues with the import process or AI agents
- Join our community forums to connect with other users
- Handle retries and timeouts gracefully

10. Support & Resources

- OmniDim support/contact: mitra@omnidim.io
- Exotel Support: hello@exotel.com/808 8919 888

5. Voice AI Ecosystem

5.1. Exotel SIP trunk API Reference for Voice AI

Use these snippets from the support articles for **ElevenLabs**, **LiveKit**, **Retell**, **Bolna**, **Pipecat (via Daily SIP)**, **Smallest AI (Atoms)**, **Vocallabs**, **Rapida AI**, and **NLPearl.AI** integrations. Replace placeholders; do not commit real secrets.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

Placeholder	Description
Account Creation	**Exotel Indian Account**
API_KEY	Exotel API Key **(API Settings**)
API_TOKEN	Exotel API Token
ACCOUNT_SID	Exotel Account SID
SUBDOMAIN	api.in.exotel.com (India) or api.exotel.com (Singapore)
TRUNK_SID	Returned when you create a trunk (trunk_sid in response)

Authentication: HTTP Basic – https://API_KEY:API_TOKEN@SUBDOMAIN/...

Trunk API rate limit: 200 requests per minute on these trunk configuration APIs (Exotel SIP API reference).

Outbound call rate (default): Exotel typically allows **200 outbound call attempts per minute** by default (account-level). If you need a higher outbound initiation limit, contact your **CSM**.

Where to get API credentials: API Settings (India) (or your cluster dashboard).

Headers: Content-Type: application/json on POST/PUT.

Exotel edge: IP and port

Exotel documents **SIP signaling toward their gateway** using **edge IP addresses and ports** – for example **TLS on port 443** or **TCP on port 5070** – per Exotel network and firewall. Use the **IP:port** (and transport) Exotel gives you in onboarding or support.

Exotel may also share **edge hostnames** (instead of raw IPs). Common India examples you may see are:

- **TCP:** in.voip.exotel.com:5070
- **TLS:** in.voip.exotel.com:443

Use the exact **host/IP + port + transport** Exotel assigns to your account/cluster. If a provider UI requires an **IPv4 literal** (no hostname), resolve the hostname to the IPv4 Exotel intends you to use (or request the explicit edge IPs from Exotel support).

The **domain_name** field in **Create trunk** is Exotel's **account SIP domain** for the trunk record ({ACCOUNT_SID}.pstn.exotel.com). That is separate from the **edge IP:port** your Voice AI platform uses to send SIP to Exotel.

Outbound trunk vs inbound SIP trunk

Flow	Exotel API steps
Outbound SIP (Voice AI → Exotel → PSTN)	**1.** Create trunk → **2.** Map DID → **3.** POST …/credentials (digest). **Optional:** POST …/whitelisted-ips **only** if the Voice AI provider gives you a **fixed static egress IP** to allow – see below. **Do not** use destination URI for this path unless Exotel instructs you otherwise.
Inbound SIP (PSTN → Exotel → Voice AI)	After trunk + DID, **map destination URI** on the trunk to the partner's SIP host/port/transport. Use **Flow** → **Connect** with **sip:<trunk_sid>** in **Dial whom** (see support articles).

Trunk ACL (whitelisted-ips)

Exotel trunk ACL is for **static IP allowlisting** from your Voice AI provider when they give you a **single fixed egress IP**. **Exotel does not support CIDR range whitelist on the trunk** – use **one IP per POST** with mask: 32. If the provider only offers non-static or range-based egress, use **digest credentials** and coordinate with Exotel on what is supported.

Trunk ACL (whitelisted-ips)

Exotel trunk ACL is for **static IP allowlisting** from your Voice AI provider when they give you a **single fixed egress IP**. **Exotel does not support CIDR range whitelist on the trunk** – use **one IP per POST** with mask: 32. If the provider only offers non-static or range-based egress, use **digest credentials** and coordinate with Exotel on what is supported.

Important (auth precedence): Avoid mixing **IP allowlisting** and **digest auth** unless Exotel support has confirmed the expected behavior for your account. In some SIP deployments, once an **IP allowlist** is enabled, the platform may treat **source-IP** as the primary trust signal, and digest behavior can become confusing (especially if a provider publishes **shared / multi-tenant** egress ranges). If your provider only publishes **CIDR ranges** (no dedicated static /32 IPs), do **not** attempt to "whitelist the range" on Exotel – rely on **digest** and work with Exotel/provider support for the correct model.

Recommendation

```
Use Credential-based (Digest) Auth for SIP outbound trunk configuration with all Voice AI providers. IP-based whitelisting should not be used as the sole or primary authentication mechanism for these trunks.
```

Create trunk

POST /v2/accounts/{ACCOUNT_SID}/trunks

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks" \
  -H "Content-Type: application/json" \
  -d '{
    "trunk_name": "my_trunk_name",
    "nso_code": "ANY-ANY",
    "domain_name": "'"${ACCOUNT_SID}"'.pstn.exotel.com"
  }'
```

trunk_name: alphanumeric + underscores, max 16 characters.

Map phone number (DID) to trunk

POST /v2/accounts/{ACCOUNT_SID}/trunks/{TRUNK_SID}/phone-numbers

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/phone-numbers" \
  -H "Content-Type: application/json" \
  -d "{\"phone_number\": \"+9198XXXXXXXX\"}"
```

Optional: "mode": "flow" for StreamKit/Voice AI routing. Save the returned ID for updates.

Create SIP digest credentials (outbound auth)

POST /v2/accounts/{ACCOUNT_SID}/trunks/{TRUNK_SID}/credentials

Use the **same** `user_name` and `password` on the Voice AI platform (ElevenLabs SIP digest, LiveKit authUsername / authPassword, Retell termination, etc.).

Curl

```
curl -s -X POST \
  "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/credentials" \
  -H "Content-Type: application/json" \
  -d '{
    "user_name": "SIP_USER",
    "password": "SIP_PASS",
    "friendly_name": "voice_ai_platform"
  }'
```

Whitelist IP (ACL) — static provider IP only

POST /v2/accounts/{ACCOUNT_SID}/trunks/{TRUNK_SID}/whitelisted-ips

Use **only** when your Voice AI provider gives a **single static egress IP** you must allow. **Repeat the call for each distinct static IP** (no CIDR ranges on trunk).

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/whitelisted-ips" \
  -H "Content-Type: application/json" \
  -d '{"ip": "203.0.113.50", "mask": 32}'
```

Set destination URI (inbound SIP — PSTN toward Voice AI partner)

POST /v2/accounts/{ACCOUNT_SID}/trunks/{TRUNK_SID}/destination-uris

Used for **inbound** routing: PSTN → Exotel → your SIP partner (FQDN or host/port/transport per partner docs). **Not** part of the minimal **outbound** SIP setup.

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "partner.example.com:5061;transport=tls" }
    ]
  }'
```

Verify (GET)

Curl

```
curl -s "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}"
curl -s "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/phone-numbers"
curl -s "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/whitelisted-ips"
curl -s "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris"
```

5.2. Connect Exotel SIP Trunk to ElevenAgents by ElevenLabs

This guide explains how to connect **Exotel SIP trunking** to **ElevenAgents by ElevenLabs** so you can place and receive PSTN calls in India through Exotel while your agent runs on ElevenLabs.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

What you will set up

Integration	Direction	When to use
Outbound SIP (digest)	ElevenLabs → Exotel → PSTN	Agent **places outbound** calls via your Exotel DID
Inbound SIP	PSTN → Exotel DID → ElevenLabs	Callers **dial** your Exotel number and reach the agent

Separate the API work: Outbound SIP uses only **create trunk → map DID → credentials**. **Inbound SIP** adds **destination URI** on the trunk (and Flow). **Trunk ACL** (whitelisted-ips) applies **only** when your Voice AI provider gives you a **static egress IP** – Exotel trunk **does not support CIDR range** allowlisting; use **one static IP per API call** (mask: 32) if needed.

Architecture

Outbound SIP (digest)

```
ElevenLabs → SIP digest → Exotel edge IP:port → Indian PSTN → Customer
```

Inbound SIP

```
Customer → PSTN → Exotel DID → Flow (Connect) → sip:<trunk_sid> → … → sip.rtc.elevenlabs.io → ElevenLabs Agent
```

Destination URI on the trunk defines where Exotel sends **inbound** SIP toward ElevenLabs. In the **Connect** applet **Dial whom**, use **sip:<trunk_sid>** only – the **trunk_sid** string returned from **Create trunk** (not a full sip:user@host URI).

For FQDN-only inbound (no digest on trunk), **do not** add POST …/credentials unless Exotel requires it for your design.

Prerequisites

Exotel

- my.in.exotel.com with **SIP trunking** enabled.
- **KYC**; Exophone (DID) in **E.164** (+91…).
- **API Key**, **API Token**, **Account SID** from API credentials.
- APIs: **https://api.in.exotel.com** (India).

ElevenLabs

- Agents / Conversational AI and **SIP** phone import.
- **Published agent**; **API key** if testing outbound via HTTP API.

Network

- SIP: **TCP** or **TLS**; RTP: **UDP** per Exotel network doc.

Part A — ElevenLabs (dashboard)

Outbound SIP (digest)

1. **Agents** → create/publish agent.
2. **Phone Numbers** → **Import a phone number from SIP trunk**.
3. **E.164** Exotel DID; **Transport** TCP/TLS per Exotel; **SIP digest** = same values you will set in Exotel POST …/credentials.
4. **Outbound address** = **Exotel edge IP:port** from Exotel.
5. Import → **assign** number to agent. Note **agent_phone_number_id** for API tests.

Inbound SIP (ACL path)

1. Import number; leave digest **empty** if using ACL on ElevenLabs.
2. Allowlist **Exotel signaling IPs** on ElevenLabs if required (Exotel network doc).

Part B — Exotel console

- Numbers, KYC, App Bazaar / Flows for **Connect** when using inbound.

Part C — Exotel APIs

Auth: https://API_KEY:API_TOKEN@api.in.exotel.com/…
Rate limit: 200 requests/minute.
Templates: _exotel-trunk-api-snippets.md.

Outbound SIP — required steps only

1. **Create trunk**
2. **Map DID** to trunk
3. **POST …/credentials** — digest; must match ElevenLabs import

Optional (only if ElevenLabs gives a static egress IP): POST …/whitelisted-ips with that **single IP**, mask: 32. Repeat per static IP if Exotel and your provider agree. **Do not** assume CIDR range whitelist on trunk.

Curl

```
curl -s -X POST \
  "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/credentials" \
  -H "Content-Type: application/json" \
  -d '{
    "user_name": "SIP_USER",
    "password": "SIP_PASS",
    "friendly_name": "eleven_labs"
  }'
```

Applicability: UI-driven (ElevenLabs console for agents + phone numbers) with optional **API-driven** outbound triggering.

Exotel edge: Exotel provides **SIP edge IP address(es) and port(s)** for signaling (for example **TLS 443** or **TCP 5070** per Exotel network and firewall). Configure **<EXOTEL_EDGE_IP>:<PORT>** in ElevenLabs — use the values Exotel assigns for your account.

Edge hostnames you may see (India): in.voip.exotel.com:5070 (TCP) and in.voip.exotel.com:443 (TLS). Use the exact host/IP + port + transport Exotel assigns. See _exotel-trunk-api-snippets.md for details.

ACL vs digest (important): Avoid trying to whitelist **CIDR ranges**. Exotel trunk ACL is intended for **static /32 IPs** only (mask: 32). If a provider publishes only **CIDR** / shared egress ranges, prefer **digest** and coordinate with Exotel/provider support—mixing allowlists and digest can cause auth/routing issues in multi-tenant egress setups.

Inbound SIP — destination URI on trunk

Map **inbound** SIP toward ElevenLabs (host/port/transport per ElevenLabs):

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "sip.rtc.elevenlabs.io:5060;transport=tcp" }
    ]
  }'
```

Connect applet (inbound)

1. App Bazaar → Flow → **Connect** applet.
2. **Dial whom: sip:<trunk_sid>** – paste the **trunk_sid** from the create-trunk API response (prefix sip: only; **not** a full SIP URI).
3. Map the Exophone to this Flow.

Overview: Exotel Voice AI / SIP trunking.

Test calls

Lab validation (outbound)

Outbound SIP has been **tested** end-to-end when trunk + DID + digest + (optional) static-IP ACL align. A **connected call** does not guarantee the **agent speaks** – see below.

Outbound API

Use ElevenAgents by ElevenLabs **outbound call** API – confirm the current URL in ElevenLabs API docs (paths change over time).

Curl

```
curl -s -X POST "https://api.elevenlabs.io/v1/convai/<outbound-call-endpoint-per-current-docs>" \
  -H "xi-api-key: ${ELEVENLABS_API_KEY}" \
  -H "Content-Type: application/json" \
  -d '{
    "agent_id": "YOUR_AGENT_ID",
    "agent_phone_number_id": "YOUR_PHONE_NUMBER_ID",
    "to_number": "+91XXXXXXXXXX"
  }'
```

Inbound

Dial your Exotel DID; agent should answer via ElevenLabs.

Call connects but agent does not speak (no bot audio)

This section applies when the **call is triggered** but the **agent does not play** audio.

Check	Action
Assignment	Imported DID **assigned** to the agent used in the API
IDs	agent_phone_number_id matches the imported SIP number
Agent	Published; first message / voice configured
Logs	ElevenLabs ConvAI traces for session errors
RTP	If total silence, check UDP/RTP per Exotel network doc

Troubleshooting

Symptom	Likely cause
SIP **401**	Digest mismatch (Exotel /credentials vs ElevenLabs)
SIP **403** inbound	ElevenLabs ACL vs Exotel source IPs
408 / timeout	DNS/firewall to ElevenLabs FQDN
Connect does nothing	**Dial whom** must be **sip:<trunk_sid>**, not a full URI
Call triggers, no speech	See Call connects but agent does not speak (no bot audio)

Official references

Resource	URL
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference
Exotel + ElevenLabs	https://docs.exotel.com/dynamic-sip-trunking/elevenlabs-and-exotel-sip-trunking-integration-guide-for-voice-ai
Exotel network	https://docs.exotel.com/dynamic-sip-trunking/network-and-firewall-configuration
ElevenLabs SIP	https://elevenlabs.io/docs/agents-platform/phone-numbers/sip-trunking

5.3. Connect Exotel SIP trunking to Pipecat (via Daily)

This guide aligns **Exotel SIP trunking** with **Pipecat** using the same **Exotel** patterns as the other Voice AI integrations in this repository: **outbound** = create trunk → map DID → credentials; **optional ACL** = static IPs only (mask: 32), no CIDR ranges on the trunk; **inbound** = destination URI on the trunk when routing toward a **fixed** SIP partner; **Connect** = **sip:<trunk_sid>** where Exotel's product uses that form.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

```
Applicability: API/engineering-driven (Pipecat orchestration + Daily rooms/SIP). Inbound is typically a dynamic
Daily sip_uri bridge, not a single static destination URI.
```

Pipecat is not a SIP trunk provider. Pipecat agents typically use **Daily** for **WebRTC** and **SIP dial-in / dial-out**. Pipecat publishes a **PSTN + Daily SIP walkthrough** (upstream doc path). Treat **Exotel** as the **PSTN carrier**: your **webhook server** still creates a **Daily room with SIP** and bridges the live call to Daily's **sip_uri** after **on_dialin_ready**.

What Pipecat + Daily expect

Topic	Detail
SIP on rooms	**Daily SIP** – sip_uri on the room, dialin-ready / Pipecat **on_dialin_ready**
PSTN example	**Pipecat PSTN + Daily SIP guide** – webhook → Daily room → forward to **sip_endpoint**
PSTN without Exotel	**Daily PSTN** – Daily-provisioned numbers

What you will set up

Direction	Summary
Inbound (typical)	PSTN → Exotel → **your webhook** → Daily room + SIP → **bridge** to Daily **sip_uri** (same **shape** as Pipecat's PSTN guide; use **Exotel** call-control APIs to attach the live leg to **sip_uri**)
Outbound	Daily **SIP dial-out** / start_dialout → **Exotel edge IP:port** (digest) → PSTN – **Exotel** trunk + DID + **POST …/credentials**

Part A – Pipecat + Daily

1. Follow **Pipecat's PSTN + Daily SIP guide** to implement **DailyTransport**, **on_dialin_ready**, and bridging to **sip_endpoint**.
2. Replace the **sample carrier** control APIs from that guide with **Exotel's** documented way to **connect the active carrier leg** to an **external SIP URI** (your Daily sip_uri for that session). Confirm details with **Exotel** if needed.
3. For **outbound**, use Daily's **dial-out** / **SIP dial-out** toward **Exotel** as the next SIP hop, then complete **Part B**.

Note: Daily exposes a **per-room** sip_uri. The **static trunk destination** pattern (single SIP host for all calls) matches some Voice AI platforms but **differs** from typical **Daily** room SIP – prefer the **dynamic bridge** unless you operate an **SBC** or fixed ingress.

Part B – Exotel (outbound SIP toward PSTN)

When **Daily** (or your SIP client) sends media to **Exotel** to reach PSTN:

1. **Create trunk**
2. **Map DID** – POST .../trunks/{TRUNK_SID}/phone-numbers
3. **POST .../credentials** – user_name, password (digest aligned with the SIP client leg)

Optional: POST .../whitelisted-ips **only** if Daily (or your egress) publishes a **single static IP** you must allow. Use **mask: 32**.

Part C – Exotel (inbound PSTN toward a fixed SIP host)

Use **only** when your **partner SIP** destination is **fixed** (not the usual **per-room Daily sip_uri** case):

1. **POST .../destination-uris** on the trunk toward that host/port/transport.
2. **Flow** → **Connect** applet: **Dial whom** = **sip:<trunk_sid>** (the **trunk_sid** from Exotel **create trunk** – not a full SIP URI).
3. Map the Exophone to the Flow.

Exotel API snippets

Authentication: https://API_KEY:API_TOKEN@api.in.exotel.com/...
Rate limit: 200 requests/minute on trunk configuration APIs.

Exotel SIP trunk API Reference for Voice AI

Troubleshooting

Symptom	What to check
Call never reaches Daily	Exotel **bridge** to the **current** sip_uri; room **SIP** enabled; **dialin-ready** / on_dialin_ready
One-way audio / codec mismatch	**SDP** codecs – Daily **PCMU/PCMA** vs Exotel PSTN leg
Outbound fails	Trunk **credentials**; **Exotel edge IP:port** and transport; DID on trunk
Connect misrouted	**sip:<trunk_sid>** only in **Dial whom** (when using that applet pattern)

Official references

Resource	URL
Daily dashboard	https://dashboard.daily.co/
Pipecat – PSTN + Daily SIP	https://docs.pipecat.ai/guides/telephony/twilio-daily-sip
Pipecat – Daily PSTN	https://docs.pipecat.ai/guides/telephony/daily-pstn
Daily – SIP	https://docs.daily.co/guides/products/dial-in-dial-out/sip
Pipecat phone example	https://github.com/pipecat-ai/pipecat-examples/tree/main/phone-chatbot
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

5.4. Connect Exotel SIP Trunk to LiveKit

This guide connects **Exotel SIP trunking** to **LiveKit Cloud** telephony so PSTN calls can reach **LiveKit rooms** and outbound calls can use Exotel as the Indian PSTN leg.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

What you will set up

Direction	Path
Inbound PSTN → LiveKit	PSTN → Exotel DID → trunk **destination URI** → LiveKit SIP endpoint → dispatch → room
Outbound PSTN ← LiveKit	LiveKit outbound trunk → digest → Exotel edge **IP:port** → PSTN

Exotel trunk ACL: use **whitelisted-ips only** when LiveKit (or your network) provides a **fixed static egress IP** to allow. Exotel trunk **does not support CIDR range** allowlisting – use **mask: 32** and **one POST per static IP**. Otherwise rely on **digest** (POST …/credentials).

Architecture

```
Inbound:  PSTN → Exotel DID → destination URI on trunk → LiveKit SIP host → room → agent
Outbound: LiveKit → Exotel edge IP:port (digest) → PSTN
```

Prerequisites

- Exotel: KYC, DID **E.164**, API credentials, https://api.in.exotel.com.
- LiveKit: Cloud project, Telephony, SIP URI from **Project settings** (SIP trunk setup).

Part A – LiveKit Cloud

SIP endpoint and region (inbound)

Copy **SIP URI** from project settings. For India pinning: {subdomain}.india.sip.livekit.cloud (region pinning).

Inbound trunk

Telephony → SIP trunks → Inbound – include your Exotel DID in E.164 (Inbound trunk).

Dispatch rule

At least one rule so calls land in a room (**Dispatch rule**).

Outbound trunk

- **address** – Exotel **edge IP:port** from Exotel.
- **numbers** – Exotel DID **E.164**.
- **authUsername / authPassword** – same as Exotel **POST …/credentials**.

JS

```
{
  "name": "Exotel outbound",
  "address": "YOUR_EXOTEL_EDGE_IP:443",
  "numbers": ["+9198XXXXXXXX"],
  "authUsername": "SIP_USER",
  "authPassword": "SIP_PASS"
}
```

Agent in the same room

OUTBOUND-EXOTEL-NOTES.md – ring without bot audio usually means no publisher in the room.

Part B – Exotel APIs

Rate limit: 200/minute. Snippets: _exotel-trunk-api-snippets.md.

Outbound SIP – three steps

1. **Create trunk**
2. **Map DID** to trunk
3. **POST .../credentials** (user_name, password) – must match LiveKit outbound trunk auth

Optional: POST .../whitelisted-ips **only if** LiveKit gives you a **static SIP egress IP** to allow – **single IP**, mask: 32 per entry. No CIDR range on trunk.

Inbound only: POST .../destination-uris toward your LiveKit SIP host (not required for minimal outbound-only testing).

Curl

```
curl -s -X POST \
  "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/credentials" \
  -H "Content-Type: application/json" \
  -d '{
    "user_name": "SIP_USER",
    "password": "SIP_PASS",
    "friendly_name": "livekit"
  }'
```

Inbound SIP – destination URI on trunk

Curl

```
curl -s -X POST "https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "YOUR_SUBDOMAIN.india.sip.livekit.cloud:5061;transport=tls" }
    ]
  }'
```

Connect applet (inbound)

Dial whom: sip:<trunk_sid> – use the **trunk_sid** from create trunk (**not** a full SIP URI). Map DID to Flow. **Detailed SIP Trunking API Reference**

Testing

Outbound SIP was validated with correct **edge IP:port**, **digest**, and **E.164**; optional static-IP ACL when applicable. Audio still requires an **agent** in the **same** room as the SIP participant.

Applicability: UI-driven + developer-driven (LiveKit Cloud console for trunks/dispatch; your app/agent joins rooms). Not a single “import trunk” wizard like some providers.

Exotel edge: Signaling toward Exotel uses **edge IP:port** from Exotel (network and firewall). Configure **IP:port** as Exotel assigns — not an assumed carrier hostname.

Edge hostnames you may see (India): in.voip.exotel.com:5070 (TCP) and in.voip.exotel.com:443 (TLS). Use the exact host/IP + port + transport Exotel assigns. See _exotel-trunk-api-snippets.md.

ACL vs digest (important): Exotel trunk ACL (whitelisted-ips) is intended for **static /32 IPs** only (mask: 32), not CIDR ranges. If the provider/network only has **CIDR/shared egress**, prefer **digest** and coordinate with Exotel support—IP allowlisting can become the primary trust signal and cause intermittent auth/routing issues in shared egress setups.

Scope: LiveKit **Cloud** with **Telephony** enabled. See OUTBOUND-EXOTEL-NOTES.md for 403, E.164, and “no audio” patterns.

Troubleshooting

Symptom	What to check
403 / 4001	Digest; optional static-IP ACL; correct **edge IP:port**
Wrong format	**E.164** for India
Ring, no bot	Agent + room (OUTBOUND-EXOTEL-NOTES.md)
Connect broken	**sip:<trunk_sid>** only in Dial whom

References

Resource	URL
LiveKit Cloud	https://cloud.livekit.io/
LiveKit SIP trunk setup	https://docs.livekit.io/telephony/start/sip-trunk-setup/
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

5.5. Connect Exotel SIP trunking to Smallest AI (Atoms)

This guide connects **Exotel SIP trunking** to **Smallest AI Atoms** using **Import SIP** (bring your own number over SIP). Smallest documents this in the platform **Phone Numbers** flow – you provide a **SIP Termination URL** (where Atoms sends **outbound** SIP toward your carrier) and copy the **SIP Origination URL** that Atoms gives you into your carrier for **inbound** routing.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

Applicability: UI-driven (Atoms “Import SIP” screen) with optional **API-driven** outbound call triggering.

Exotel edge: Use **IP:port** (and transport) from Exotel for SIP toward their gateway (network and firewall). Enter the value Smallest expects as the **termination** target (often host:port or sip:host:port — follow the **Import SIP** form and phone numbers doc).

Edge hostnames you may see (India): in.voip.exotel.com:5070 (TCP) and in.voip.exotel.com:443 (TLS). Use the exact host/IP + port + transport Exotel assigns. See _exotel-trunk-api-snippets.md.

ACL vs digest (important): Exotel trunk ACL (whitelisted-ips) is intended for **static /32 IPs** only (mask: 32). If a provider publishes only **CIDR/shared egress**, do **not** attempt to whitelist ranges—prefer **digest** and coordinate with Exotel/provider support if calls fail.

Origination URL is tenant-specific: After you add a custom SIP number, Atoms shows a **SIP Origination URL** — **copy it exactly** into Exotel **destination-uris** for PSTN → Exotel → Smallest. Do **not** reuse example hostnames from third-party summaries.

Engineering detail: smallest/integrations/exotel-vsip/smallest-exotel-voice-ai-connector.md

Quickstart: smallest/integrations/exotel-vsip/QUICKSTART.md

Smallest AI (from official docs)

Topic	Detail
Import SIP	**Phone Numbers – Import SIP**: **Phone Number** (E.164), **SIP Termination URL**, optional **Username** / **Password**, **SIP Origination URL** (provided by Atoms → paste into provider)
Telephony overview	**Phone Numbers capability** – inbound, outbound, **SIP** to existing infrastructure
Outbound API	**Start an outbound call** – requires agent + linked phone number
Console	app.smallest.ai

Codec / security: Smallest's public docs describe SIP integration in product terms; confirm **codec** (e.g. G.711 / Opus) and **SRTP** requirements with **Smallest support** if calls fail at SDP negotiation.

Flows

Direction	What to configure
Outbound SIP (Atoms → Exotel → PSTN)	**Exotel:** create trunk → map DID → **POST .../credentials** (digest). **Smallest: Import SIP** – **SIP Termination URL** = Exotel **edge IP:port** (format per UI); **Username** / **Password** = same as Exotel credentials if you use digest auth.
Inbound SIP (PSTN → Exotel → Atoms)	**Smallest:** copy **SIP Origination URL** from the imported number. **Exotel: POST .../destination-uris** on the trunk with that URI (host, port, ;transport= per Atoms). **Flow** → **Connect: sip:<trunk_sid>** in **Dial whom**.

Exotel trunk ACL (whitelisted-ips): use **only** if Smallest publishes **fixed static SIP egress IPs** you must allow – **one IP per POST**, **mask: 32**. Exotel trunk **does not** support arbitrary CIDR range ACL entries. If Smallest does not publish static egress IPs, rely on **digest** and Smallest's documented networking.

Part A – Smallest AI (Atoms)

1. Sign in at app.smallest.ai.
2. **Phone Numbers** → **Add Number** → **Import SIP** (**guide**).
3. Enter your Exotel DID in **E.164**.
4. **SIP Termination URL:** Exotel **edge** host/IP and port from **Exotel network doc** (align **transport** with Exotel and the Import SIP field).
5. If using SIP digest with Exotel, set **Username** / **Password** to match **POST .../credentials** on the Exotel trunk.
6. Save and copy **SIP Origination URL** for Exotel **inbound** routing.
7. Assign the number to your agent (**Agent Settings** → **Phone Number** per **docs**).

Part B – Exotel APIs

Auth: API_KEY:API_TOKEN@api.in.exotel.com · **200 requests/minute (SIP trunk APIs)** · _exotel-trunk-api-snippets.md

Outbound SIP (minimal)

1. Create trunk
2. Map DID

3. POST …/credentials – must match Smallest **Username** / **Password** when used

Optional: POST …/whitelisted-ips only for each **static** Smallest egress IP Smallest documents (mask: 32).

Inbound SIP – destination URI on trunk

Use the **SIP Origination URL** from Atoms (example shape only – **replace with your copied value**):

```
curl -s -X POST
"https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "<PASTE_SIP_ORIGINATION_URI_FROM_ATOMS>" }
    ]
  }'
```

Connect applet (inbound)

Dial whom: sip:<trunk_sid> – **Detailed SIP Trunking API Reference**

Testing

- **Outbound: Start an outbound call** with agentId and the imported **phone number**.
- **Inbound:** PSTN dial your Exotel DID; call should reach the agent assigned in Atoms.

Troubleshooting

Symptom	What to check
401 on SIP	Digest mismatch (Exotel /credentials vs Smallest Import SIP)
408 / timeout	**Termination URL** wrong **IP:port** or transport blocked
Inbound never reaches Atoms	**destination-uris** must match **SIP Origination URL** exactly; **Connect** uses **sip:<trunk_sid>**
Media / codec issues	Confirm codecs and SRTP with Smallest docs or support

References

Resource	URL
Smallest – app	https://app.smallest.ai/
Atoms – Phone Numbers (Import SIP)	https://atoms-docs.smallest.ai/platform/deployment/phone-numbers.md
Atoms – Telephony capability	https://atoms-docs.smallest.ai/intro/capabilities/telephony.md
Atoms – Start outbound call API	https://atoms-docs.smallest.ai/api-reference/calls/start-an-outbound-call.md
Atoms – Get acquired phone numbers	https://atoms-docs.smallest.ai/api-reference/phone-numbers/get-acquired-phone-numbers.md
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

5.6. Connect Exotel SIP trunking to NLPearl.AI

This guide connects **Exotel SIP trunking** to **NLPearl.AI** using NLPearl's **Custom VoIP** feature, where **Exotel is the SIP carrier** behind NLPearl (NLPearl's portal drives the AI agent; Exotel provides the PSTN DID and SIP trunk).

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

Applicability: UI-driven + API-driven (Custom VoIP configuration in NLPearl portal; optional outbound via API).

Edge hostnames you may see (India): in.voip.exotel.com:5070 (TCP) and in.voip.exotel.com:443 (TLS). Use the exact host/IP + port + transport Exotel assigns. See _exotel-trunk-api-snippets.md.

NLPearl (from official docs)

Topic	Detail
Custom VoIP	**Custom VoIP integration** – inbound SIP Domain shown after save; outbound requires SIP Trunk URL + User Part + optional auth
Getting started	Getting started
Outbound API	**Outbound**/API and **Make Call API request**
Variables / callData	**Variables** – callData used in Make Call / Lead APIs

Flows (what you configure)

Direction	What to configure
Outbound (NLPearl → Exotel → PSTN)	**Exotel:** trunk + DID + **digest credentials**. **NLPearl:** Custom VoIP **Outbound** with **SIP Trunk URL** (Exotel trunk domain) + **User Part** + optional **credentials auth**.
Inbound (PSTN → Exotel → NLPearl)	**NLPearl:** Custom VoIP **Inbound** (choose IP auth or credentials) → save to get a **SIP Domain**. **Exotel:** set trunk **destination-uris** to that NLPearl SIP Domain (plus transport/port). **Exotel Flow:** Connect using **sip:<trunk_sid>**.

Important: **Outbound** does **not** need destination-uris. **Inbound** does.

Part A – NLPearl portal (Custom VoIP)

A1. Prepare the AI agent

1. Create and **publish** your agent (Pearl) in platform.nlpearl.ai.
2. Ensure the agent supports your required language/flow and is ready for inbound/outbound usage.

A2. Add Custom VoIP phone number

1. Open **Settings** → **Phone Numbers**.
2. Click **Custom VoIP**.
3. Enter your **Exotel DID** in E.164 (display/reference).
4. Choose **Call direction**: inbound, outbound, or both.

A3. Configure Outbound (Exotel as SIP trunk)

In the **Outbound configuration** section (**Custom VoIP docs**):

- **TLS (SRTP) Encryption**: enable only if you will use **SIP TLS** to Exotel and Exotel confirms TLS/SRTP requirements for your account.
- **SIP Trunk URL**: set to Exotel trunk SIP domain:
  - sip:${ACCOUNT_SID}.pstn.exotel.com (recommended form), or
  - ${ACCOUNT_SID}.pstn.exotel.com if NLPearl UI expects host-only.
- **User Part**: use one of:
  - your Exotel trunk **digest username**, or
  - your DID in E.164 (if you want the SIP From-user to mirror the DID).
- **Authentication methods (Credentials Authentication)**: if enabled, set the **same** username/password as Exotel **POST …/credentials**.
- **Data center**: pick the closest region to Exotel's SIP edge for latency.

A4. Configure Inbound (NLPearl receives from Exotel)

In the **Inbound configuration** section (Custom VoIP docs):

- **TLS (SRTP) Encryption**: enable only if you will use TLS/SRTP and Exotel confirms.
- **Authentication**:
  - **Credentials Authentication**: recommended when you don't have stable IPs and want strict auth.
  - **IP Address Authentication**: only if Exotel provides the fixed IPs NLPearl should accept traffic from.

After saving inbound settings, NLPearl shows a **SIP Domain to connect**. You will copy that domain into Exotel **destination-uris** (Part B).

Part B — Exotel APIs

Auth: API_KEY:API_TOKEN@api.in.exotel.com · **200 requests/minute (SIP trunk APIs)** · _exotel-trunk-api-snippets.md

B1. Create trunk + map DID + set digest credentials (outbound prerequisite)

Use the shared snippets:

- Create trunk
- Map phone number (DID)
- POST …/credentials

For NLPearl **Outbound**, these digest credentials should match what you configure in NLPearl's **Outbound** auth section.

Exotel SIP trunk API Reference for Voice AI

B2. Inbound routing: set destination URI toward NLPearl SIP Domain

Once NLPearl gives you a **SIP Domain**, set it as the trunk destination URI:

```
curl -s -X PUT
"https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "YOUR_NLPEARL_SIP_DOMAIN:5061;transport=tls" }
    ]
  }'
```

Notes:

- Use **the exact domain** NLPearl shows after saving inbound config.
- Align **port + transport** with how you configured NLPearl (TLS vs TCP). If you're unsure, start with what Exotel supports for your SIP trunking and what NLPearl's Custom VoIP expects.

B3. Exotel Flow: Connect applet

In your Exotel flow, use:

- **Applet**: Connect
- **Dial whom**: sip:<trunk_sid>

(trunk_sid is the value returned by create trunk; it is not a SIP URI.)

Tests (quick)

Outbound (NLPearl → PSTN)

1. In NLPearl, create/assign an outbound activity using the Custom VoIP number.
2. Trigger a test call from the UI and confirm Exotel sees SIP invites on the trunk.
3. If using API-driven outbound, use the Make Call endpoint per docs: **Make Call API request.**

Inbound (PSTN → NLPearl)

1. Call the Exotel DID from a mobile phone.
2. Verify Exotel routes to trunk (Flow Connect sip:<trunk_sid>) and then to the trunk destination-uris (NLPearl SIP Domain).
3. Confirm NLPearl receives the call and the assigned agent answers.

Troubleshooting

Symptom	Likely cause	Fix
401/403 on outbound SIP	Digest mismatch	Ensure NLPearl outbound credentials match Exotel POST …/credentials exactly
Inbound reaches Exotel but not NLPearl	Missing/wrong destination-uris	Paste NLPearl SIP Domain exactly; confirm port/transport; re-GET destination URIs
TLS failures	TLS/SRTP toggles mismatched	Keep TLS/SRTP off on both sides for first smoke test unless Exotel confirms required TLS/SRTP; then enable consistently
Confusing auth failures after enabling allowlist	ACL + digest interaction	Remove whitelisted-ips unless you have dedicated /32 static IP requirements; rely on digest and coordinate with Exotel

5.7. Connect Exotel SIP trunking with Vocallabs (Superflow B2B API)(Alpha)

This guide links **Exotel SIP trunking** and your **Exotel DID** to **Vocallabs**. Vocallabs publishes a **B2B REST API** on **https://api.superflow.run/b2b/** – including **createSIPCall** with did, phone_number, websocket_url, and webhook_url – not the same **dashboard BYO SIP trunk** pattern as Vapi or ElevenLabs. Use this article for **API + telephony alignment**; confirm **inbound SIP URIs** and **static egress IPs** with Vocallabs if you need classic trunk **destination-uris** / **whitelisted-ips**.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

Applicability: API-driven (token + createSIPCall / websockets / webhooks). SIP trunk routing is optional and only applies if Vocallabs provides a stable SIP target.

Docs: Vocallabs API documentation (hosted API reference and samples).

Engineering detail: vocallabs/integrations/exotel-vsip/vocallabs-exotel-voice-ai-connector.md

Quickstart: vocallabs/integrations/exotel-vsip/QUICKSTART.md

Vocallabs API (from official docs)

Topic	Detail
Base	https://api.superflow.run/b2b/
Auth	POST .../createAuthToken/ – JSON clientId, clientSecret → **Bearer** token for subsequent calls (docs)
SIP call	POST .../vocallabs/createSIPCall – phone_number, did, websocket_url, webhook_url, sample_rate
Outbound-style calls	POST .../vocallabs/initiateVocallabsCall – agentId, prospect_id
Webhook dial	POST .../vocallabs/initiateCallWebhook – from (DID), number (destination)
Websocket URL	GET .../vocallabs/getWebsocketUrl – query agent_id, prospect_id

Wallet / usage: GET .../getGreenBalance, transaction history – see Vocallabs docs for billing context.

How this maps to Exotel

Concept	Exotel side	Vocallabs side
DID	Exophone on SIP trunk (snippets)	Use the same E.164 as **did** (or from in webhook flows) where the API expects your caller / rented number
PSTN termination	Exotel edge **IP:port** + digest (POST …/credentials)	**Not** fully specified in the public reference as a single "SIP gateway hostname" like other providers – **coordinate with Vocallabs** for SIP leg details if createSIPCall is meant to drive traffic to Exotel
Inbound PSTN → AI	Trunk **destination-uris** + Flow **Connect sip: <trunk_sid>**	Requires a **SIP origination target** from Vocallabs. If their team only documents **REST + websocket**, implement that path first; add **destination-uris** when they provide a stable SIP URI

Part A – Vocallabs (auth + SIP call)

1. Create token

```
curl -s -X POST 'https://api.superflow.run/b2b/createAuthToken/' \
  -H 'Content-Type: application/json' \
  -d '{"clientId":"CLIENT_ID","clientSecret":"CLIENT_SECRET"}'
```

Use the returned token as **Authorization: Bearer <token>**.

2. Create SIP call (example shape)

Per **Vocallabs API docs**:

```
curl -s -X POST 'https://api.superflow.run/b2b/vocallabs/createSIPCall' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer AUTH_TOKEN' \
  -d '{
    "phone_number": "+91XXXXXXXXXX",
    "did": "+91YYYYYYYYYY",
    "websocket_url": "wss://your-bridge.example.com/media",
    "webhook_url": "https://your-app.example.com/vocallabs/webhook",
    "sample_rate": "16000"
  }'
```

Replace **did** with your **Exotel number** in E.164 when that number is the identity Vocallabs should use for the session. **websocket_url** / **webhook_url** must be **your** endpoints as required by Vocallabs – see their latest field definitions.

3. Other flows

- **initiateVocallabsCall** – agent + prospect (docs).
- **initiateCallWebhook** – from / number for click-to-call style flows.
- **getWebsocketUrl** – retrieve media URL for agent_id + prospect_id.

Part B – Exotel (SIP trunk + DID)

When you use Exotel as the **Indian PSTN carrier** for the same DID you pass to Vocallabs:

1. **Create trunk** → **map DID** → **POST …/credentials** (if your design uses SIP digest toward Exotel).

2. Templates: _exotel-trunk-api-snippets.md.
3. **Optional:** POST .../whitelisted-ips **only** if Vocallabs publishes **fixed static egress IPs** (mask: 32 per IP).
4. **Inbound** (only if Vocallabs gives you a **SIP URI** to send calls to):
   POST .../destination-uris toward that URI → **Flow** → **Connect** → **sip:<trunk_sid>** (**Detailed SIP Trunking API Reference**.

Troubleshooting

Symptom	What to check
401 on Superflow	Valid **createAuthToken** + **Bearer** header
Call fails / wrong CLI	**did** / **from** match **E.164** Exotel number you own
No media	**websocket_url**, **getWebsocketUrl**, firewall to Vocallabs / your bridge
SIP vs API mismatch	Vocallabs may be **API-first** – confirm with their team before assuming full SIP trunk parity

References

Resource	URL
Vocallabs API docs	https://docs.vocallabs.ai/vocallabs
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference
Exotel network / firewall	https://docs.exotel.com/dynamic-sip-trunking/network-and-firewall-configuration

5.8. Connect Exotel SIP trunking to Retell AI

This guide connects **Exotel SIP trunking** to **Retell AI** using elastic SIP per Retell custom telephony, with Exotel as the Indian PSTN provider.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

Applicability: UI-driven (Retell dashboard custom telephony) with optional **API-driven** call control.

Edge hostnames you may see (India): in.voip.exotel.com:5070 (TCP) and in.voip.exotel.com:443 (TLS). Use the exact host/IP + port + transport Exotel assigns. See _exotel-trunk-api-snippets.md.

ACL vs digest (important): Exotel trunk ACL is intended for **static /32 IPs** only (mask: 32), not CIDR ranges. If Retell provides **CIDR ranges** (common), do **not** attempt to whitelist ranges on Exotel—prefer **digest** and coordinate with Exotel/provider support if you see auth/routing issues.

Full reference: retell/integrations/exotel-vsip/retell-exotel-voice-ai-connector.md

Quickstart: retell/integrations/exotel-vsip/QUICKSTART.md

Retell (from official docs)

- **SIP server:** sip:sip.retellai.com with ;transport=tcp (or tls / udp) – **Custom telephony.**
- Retell publishes **IP ranges** for carriers to allowlist **toward Retell**; on **Exotel trunk ACL**, you can only add **static single IPs** (mask: 32) per Exotel – **not** full CIDR ranges on the trunk. If Retell gives ranges, coordinate with Exotel or use **digest** auth.

Flows

Direction	What to configure
Outbound SIP	Exotel: **create trunk → map DID → credentials**. Optional **whitelisted-ips** only if Retell provides a **fixed static egress IP** (one IP per POST, mask: 32).
Inbound SIP	Exotel: **destination URI** on trunk toward Retell (sip.retellai.com host/port/transport per Retell). **Connect** applet: **sip:<trunk_sid>** in **Dial whom** (value = API trunk_sid, not a full URI).

Part A – Retell

1. Configure agent; **import** Exotel DID after Exotel trunk exists (Custom telephony).
2. Match **digest** to Exotel POST …/credentials for outbound termination.

Part B – Exotel APIs

Auth: API_KEY:API_TOKEN@api.in.exotel.com · **200 requests/minute (SIP trunk APIs)** ·

Exotel SIP trunk API Reference for Voice AI

Outbound SIP

1. Create trunk
2. Map DID
3. POST …/credentials
4. *(Optional)* whitelisted-ips – **only** for a **static IP** from Retell, mask: 32

Inbound SIP

POST …/destination-uris – example shape (confirm port/transport with Retell):

```
curl -s -X POST
"https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "sip.retellai.com:5060;transport=tcp" }
    ]
  }'
```

Connect: sip:<trunk_sid> in Dial whom.

Dial to SIP URI (alternative)

Register Phone Call → dial sip:{call_id}@sip.retellai.com within **5 minutes** – **Method 2.**

References

Resource	URL
Retell dashboard	https://dashboard.retellai.com/
Retell – custom telephony	https://docs.retellai.com/deploy/custom-telephony
Retell – quick start	https://docs.retellai.com/get-started/quick-start
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

5.9. Connect Exotel SIP trunking to Rapida AI

This guide aligns **Exotel** with **Rapida AI** for voice assistants. Rapida documents **two** relevant patterns:

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

```
Applicability: Hybrid — UI-driven native Exotel integration (webhook/streaming) OR SIP-driven SIP trunk path.
```

1. **Native Exotel integration** – Exotel **app / Flow** calls Rapida's **webhook + bidirectional media stream** (no Exotel SIP trunk API required for this path).
2. **SIP trunk** – Exotel as **SIP carrier** (SIP trunking) ↔ Rapida's **SIP server** at **sip-01.in.rapida.ai:5060** – same **trunk / credentials / destination-uris** patterns as other articles in this repo.

```
Primary Rapida reference: Exotel integration · SIP trunk integration · Phone deployment
```

```
Engineering detail: rapida/integrations/exotel-vsip/rapida-exotel-voice-ai-connector.md
```

```
Quickstart: rapida/integrations/exotel-vsip/QUICKSTART.md
```

Choose a path

Path	Best when	Exotel work	Rapida work
A – Native Exotel	You want Rapida's documented **streaming** integration	Configure **App / Flow** webhook to Rapida; assign DID	**Integration** → **Tools** → Exotel credentials; **Phone** deployment with **App ID** + number
B – SIP trunk (SIP trunking)	You need **SIP** toward Rapida from Exotel (or digest toward Exotel for outbound)	**Create trunk** → **map DID** → **credentials**; **destination-uris** toward Rapida SIP host; **Connect sip:<trunk_sid>**	**SIP Trunk** credential + **Phone** deployment with **SIP** as provider

If you use **Path A**, you may not need **POST .../trunks** at all – follow Rapida's Exotel guide first.

Path A – Native Exotel (webhook + media stream)

From Rapida – Exotel:

Rapida – credentials

1. **Integration** → **Tools** → **Exotel** → **Setup Credential**.
2. Fields (per Rapida): **Account SID**, **Client ID**, **Client Secret** – sourced from the Exotel dashboard (API / app settings).
    - Exotel India often labels credentials **API Key** and **API Token**; map them to whatever Rapida's form expects (Exotel API credentials for India).

Rapida – phone deployment

1. Assistant → **Deploy** → **Phone**.
2. **Telephony:** provider **Exotel**, select credential, **Phone number** (Exotel virtual number), **App ID** (Exotel applet / flow id that will hit Rapida).

Exotel — applet / Flow

Set the inbound URL Rapida documents (replace placeholders):

```
https://websocket-01.in.rapida.ai/v1/talk/exotel/call/{your-assistant-id}?x-api-key={your-api-key}
```

Assign your Exotel number to this app. **Confirm the exact hostname and path** in Rapida's Exotel guide if it changes.

Outbound

Use Rapida **SDK** or **REST** (POST https://api.rapida.ai/v1/talk/call) — examples in Rapida — Exotel.

Path B — Exotel SIP trunking + Rapida SIP trunk

From **Rapida — SIP trunk**:

Rapida SIP endpoint (for routing to Rapida)

```
sip:sip-01.in.rapida.ai:5060
```

Transport **UDP** or **TCP**; codecs **G.711 PCMU** (preferred) / **PCMA**. Allow **RTP** per Rapida + your firewall docs.

Exotel — inbound PSTN → Rapida

1. **Create trunk** → **map DID** → (optional) **POST .../credentials** if your design requires digest on the Exotel side.
2. **POST .../destination-uris** toward Rapida, e.g.:

```
curl -s -X POST
"https://${API_KEY}:${API_TOKEN}@${SUBDOMAIN}/v2/accounts/${ACCOUNT_SID}/trunks/${TRUNK_SID}/destination-uris" \
  -H "Content-Type: application/json" \
  -d '{
    "destinations": [
      { "destination": "sip-01.in.rapida.ai:5060;transport=udp" }
    ]
  }'
```

1. **Flow** → **Connect: Dial whom** = **sip:<trunk_sid>** (Voice AI / SIP trunking).

Rapida — SIP trunk credential

Integration → **Tools** → **SIP Trunk**: set **SIP URI** to your **Exotel edge IP:port** (and digest if used) for **outbound** from Rapida toward Exotel — mirror **ElevenLabs-style** termination. Use Exotel network and firewall for edge values.

Optional Exotel ACL: POST .../whitelisted-ips **only** if Rapida publishes **fixed static SIP egress IPs** to allow — **one IP per POST**, **mask: 32** (Exotel trunk does not support arbitrary CIDR ranges).

Exotel SIP trunk API Reference for Voice AI

Troubleshooting

Symptom	What to check
Inbound never hits Rapida (Path A)	Webhook URL, **assistant id**, **x-api-key**, number linked to correct app
SIP failures (Path B)	**sip-01.in.rapida.ai** reachability, **UDP/TCP**, **codec**; **destination-uris** format
Outbound fails	Rapida credential + deployment; **from** number is your Exotel DID
Connect misrouted	**sip:<trunk_sid>** only in Dial whom (Path B)

References

Resource	URL
Rapida – Exotel	https://doc.rapida.ai/integrations/telephony/exotel
Rapida – SIP trunk	https://doc.rapida.ai/integrations/telephony/sip
Rapida – Phone deployment	https://doc.rapida.ai/voice-deployment-options/phone
Rapida – Credentials	https://doc.rapida.ai/credential/rapida-credentials
Rapida – Overview	https://doc.rapida.ai/introduction/overview
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

5.10. Connect Exotel SIP trunking to Bolna Voice AI

This guide aligns **Exotel SIP trunking** with **Bolna AI** using the same patterns as the other Voice AI integrations in this repository: **outbound Exotel** = create trunk → map DID → credentials; **inbound** = destination URI on the trunk; **Connect** = **sip:<trunk_sid>**; **ACL** = static IPs only (mask: 32), no CIDR ranges on the trunk.

GitHub repo (reference): https://github.com/exotel/AgentStream-VoiceAIEcosystem

```
Applicability: Hybrid — BYOT is API/config-driven (SIP trunk objects) and may also have portal steps depending on account access (SIP is Beta).
```

Outbound SIP is supported. With BYOT, Bolna **places outbound PSTN calls over SIP** through your trunk: it resolves from_number to your trunk, then signals to your **gateway** (Exotel **edge IP:port**) using **userpass** (or **ip-based**) auth. That is the flow described in Make Outbound Calls via Your SIP Trunk. **Part B** below is exactly what Exotel needs on the **carrier** side for that outbound SIP leg.

Bolna also offers a **dashboard connector** for Exotel (REST API link) – see Connect Your Exotel Account to Bolna. This article focuses on **SIP BYOT** where you configure **Exotel SIP trunking** and a **Bolna SIP trunk** together.

```
Bolna: SIP trunking is Beta; SRTP is not supported (plain RTP). See Bring Your Own SIP Trunk.
Engineering detail: [bolna/integrations/exotel-vsip/bolna-exotel-voice-ai-connector.md](../../bolna/integrations/exotel-vsip/bolna-exotel-voice-ai-connector.md) Quickstart: [bolna/integrations/exotel-vsip/QUICKSTART.md](../../bolna/integrations/exotel-vsip/QUICKSTART.md)
```

What Bolna expects (BYOT)

Topic	Detail
Create trunk	**Create SIP Trunk API** – POST https://api.bolna.ai/sip-trunks/trunks with Bearer token
Auth	userpass (username/password) or ip-based with IP identifiers on Bolna
Gateway	Points to your carrier – for Exotel, use **Exotel edge IP:port** from **Exotel network doc**
Inbound origination	Carrier sends traffic to **sip:13.200.45.61:5060** per **Receive Inbound Calls via Your SIP Trunk** – confirm in current Bolna docs
Outbound calls	**Make Outbound Calls via Your SIP Trunk** – agent telephony_provider: sip-trunk, from_number on trunk

What you will set up

Direction	Summary
Outbound	Bolna → SIP digest → Exotel edge → PSTN
Inbound	PSTN → Exotel DID → trunk **destination URI** toward Bolna (13.200.45.61:5060 per Bolna) → Bolna agent

Part A – Bolna

1. Obtain **SIP trunk / BYOT** access (enterprise@bolna.ai if required).

2. **Create SIP trunk** with gateway = **Exotel edge IP:port** and userpass matching Exotel POST …/credentials – this is what enables **outbound SIP** from Bolna → Exotel → PSTN.
3. **Add phone numbers** to the Bolna trunk (required for from_number on outbound calls).
4. Set agent telephony_provider to sip-trunk, then place outbound calls with the **call API** (from_number must be a DID on the trunk).
5. For **inbound** only: map numbers to agents per **inbound BYOT** and ensure Exotel routes to **sip:13.200.45.61:5060** per Bolna (Part C).

Alternative: Use **Exotel provider connectio**n in Bolna **Providers** instead of manual SIP BYOT when that product path fits.

Part B – Exotel (outbound SIP)

1. **Create trunk**
2. **Map DID** – POST …/trunks/{TRUNK_SID}/phone-numbers
3. POST …/credentials – user_name, password (same as Bolna auth_username / auth_password for userpass)

Optional: POST …/whitelisted-ips **only** if Bolna publishes a **single static IP** you must allow on Exotel (Bolna docs reference **13.200.45.61** in troubleshooting – **always confirm** with current Bolna documentation). Use **mask: 32**.

Part C – Exotel (inbound SIP)

1. POST …/destination-uris on the trunk so **inbound** PSTN is routed toward Bolna's SIP entry (host/port per Bolna – typically aligned with **13.200.45.61:5060**; format per Exotel API).
2. **Flow** → **Connect** applet: **Dial whom** = **sip:<trunk_sid>** (the trunk_sid from Exotel **create trunk** – not a full SIP URI).
3. Map the Exophone to the Flow.

Exotel API snippets

Authentication: https://API_KEY:API_TOKEN@api.in.exotel.com/…
Rate limit: 200 requests/minute on trunk configuration APIs.

Full reference: **Detailed SIP Trunking API Reference**

Troubleshooting

Symptom	What to check
No media / failed SDP	**SRTP** – Bolna does not support SRTP; use plain RTP
Outbound fails	Trunk is_active on Bolna; gateway IP:port; digest match
Inbound never hits Bolna	Exotel **destination URI**; Bolna inbound_enabled; number mapping
Connect misrouted	**sip:<trunk_sid>** only in Dial whom

Official references

Resource	URL
Bolna BYOT setup (start here)	https://www.bolna.ai/byot-setup
Bolna SIP introduction	https://www.bolna.ai/docs/sip-trunking/introduction
Bolna + Exotel (dashboard)	https://www.bolna.ai/docs/exotel-connect-provider
Bolna Create SIP Trunk	https://www.bolna.ai/docs/api-reference/sip-trunks/create
Bolna BYOT outbound	https://www.bolna.ai/docs/sip-trunking/byot-outbound-calls
Bolna BYOT inbound	https://www.bolna.ai/docs/sip-trunking/byot-inbound-calls
Exotel SIP API	https://docs.exotel.com/dynamic-sip-trunking/detailed-sip-trunking-api-reference

6. FAQs

6.1. AgentStream: what to use when

Short chooser for connecting Voice AI over **Exotel AgentStream.**

There are **two control models**:

Model	How call logic is defined	Typical entry
Flow (App Bazaar)	Dashboard applets (Voicebot, Stream, Connect, Passthru, Gather, Greeting)	Exophone → Appbazaar flow, Voicebot applet or Passthru/ Connect Applet with Url
ExoML (Programmable Voice APIs)	Your code reacts to **gRPC leg events** and issues **leg actions** (start_stream, start_say, start_play, hangup, bridge, ...). **No App Bazaar flow required.**	Number attached to gRPC endpoint (inbound), or POST /legs (outbound)

Your bot WebSocket (AgentStream) is the same either way. Choose the **telephony control plane**, not a different bot protocol.

Quick chooser

You need...	Use
Outbound: answer → bot only	**Connect Voice AI API**
Outbound: answer → bot with custom routing logics via applets: Voicebot applet and then / IVR / greeting / DTMF / agent handoff via connect applet in Exotel	**Connect Voice AI with Flow API**
Outbound or inbound: full runtime control in code, per-call stream URL, parallel greeting while bot connects	**ExoML / Programmable Voice APIs**
Inbound: dashboard flow with Voicebot applet	Assign virtual number (Exophone) to an **App Bazaar flow** that includes **Voicebot applet**
Inbound: no flow – events to your app, you drive the call	Attach virtual number to a **gRPC endpoint** (ExoML / Programmable Voice)

Outbound

1. Connect Voice AI API – simplest outbound

Docs: Connect Voice AI API

What happens: Your API dials the customer. On answer, Exotel opens bidirectional WSS to StreamUrl.

Use when:

- Campaigns, surveys, reminders – bot is the whole experience

- No IVR, no compliance applet, no agent transfer in Exotel
- Fixed bot URL is fine in the request (StreamUrl + StreamType=bidirectional)

Not for: Menus, disclosure before bot, bot→human, or per-leg programmable actions.

```
Your API → dial customer → answer → AgentStream WSS ↔ bot
```

2. Connect Voice AI with Flow API — outbound + App Bazaar journey

Docs: **Connect Voice AI with Flow API**

What happens: Same connect call, but Url points at an App Bazaar flow. After answer, applets run. When the flow hits **Voicebot** / **Stream**, Exotel opens WSS. Stream URL is configured **in the applet**, not in the API body.

Use when:

- Compliance greeting → bot
- IVR / DTMF → bot
- Bot → Connect (human) or Passthru (your system)
- Unidirectional Stream for assist / transcription

Not for: Pure "answer → bot" (use Connect Voice AI), or code-owned per-leg control (use ExoML).

```
Your API → dial customer → answer → App Bazaar flow → Voicebot/Stream → WSS ↔ bot
```

Common flow patterns: Greeting → Voicebot · Gather → Voicebot · Voicebot → Connect · Voicebot → Passthru

3. ExoML (Programmable Legs) — outbound in code

Docs: **Programmable Voice APIs with AgentStream** · **ExoML introduction**

What happens: You POST /legs to dial the customer, receive events on your **gRPC** leg_event_endpoint, then issue leg actions yourself (e.g. start_stream, optional start_say / start_play).

Use when:

- Stream URL / routing decided per call at runtime
- Parallel filler while bot connects (sales / collections "instant feel")
- You want application-owned call logic, not a dashboard flow

Scenario	Pattern
Bot-first / tech support	leg_answered → start_stream
Sales / collections / CC simulation	leg_answered → start_stream **+** short start_say / start_play → on stream_started → stop_say / stop_play

```
Your API → POST /legs → gRPC events (connecting / ringing / answered)   → your app: start_stream (+ say/play) → AgentStream WSS ↔ bot
```

Inbound

Inbound has **two separate paths**. Do not mix them with the same mental model.

A. Flow-based inbound (App Bazaar + Voicebot applet)

What happens: Customer dials your Exophone. The number is linked to an **App Bazaar flow**. Applets execute in the dashboard-defined order. **Voicebot** opens bidirectional AgentStream to your bot.

Use when:

- You want visual / applet-based journeys (Voicebot, greeting, IVR, bot, agent handoff)
- Same patterns as Connect-with-Flow outbound, but the customer dials in

Customer → Exophone → App Bazaar flow → Voicebot → WSS ↔ bot

Setup: Procure Exophone → assign **Voice URL / flow** → include Voicebot (and optional Connect / Passthru / Gather).

B. ExoML programmable inbound (no App Bazaar flow)

What happens: Customer dials your Exophone. The number is attached to a **gRPC endpoint** (Programmable Voice / Legs), **not** to an App Bazaar flow. Exotel pushes leg events to your gRPC service. **Your application is the flow** – you answer/control the leg with **leg actions** (start_stream, say, play, gather, bridge, hangup, ...).

Use when:

- Call logic must live entirely in code
- Per-call dynamic stream URL, A/B routing, CRM-driven branching
- Same ExoML action model as outbound Legs (inbound or outbound legs)

```
Customer → Exophone (attached to gRPC) → gRPC leg events → your app   → leg actions (start_stream, say/play, …) → AgentStream WSS ↔ bot
```

Setup: Implement gRPC event endpoint → register it → attach Exophone to that gRPC endpoint → on inbound events, issue leg actions.

ExoML inbound does **not** require a Voicebot applet or App Bazaar flow. The programmable “flow” is the sequence of leg actions your service sends in response to gRPC events.

Side-by-side

	Connect Voice AI	Connect + Flow	ExoML (Legs)	Inbound Flow + Voicebot	Inbound ExoML
Direction	Outbound	Outbound	Outbound **and** inbound	Inbound	Inbound
Control plane	Single connect API	App Bazaar applets	gRPC events + leg actions	App Bazaar applets	gRPC events + leg actions
App Bazaar flow	**No**	Yes	**No**	Yes	**No**
Where WSS URL is set	API StreamUrl	Voicebot/Stream applet	start_stream action	Voicebot/Stream applet	start_stream action
Parallel greeting while bot warms	Bot-owned	Flow Play applet	start_say / start_play + stop on stream_started	Bot-owned	Same ExoML say/play pattern
Best for	Simple outbound bot	Multi-step outbound	Code-first, dynamic control	Dashboard inbound journeys	Code-first inbound

Decision tree

```
Who starts the call?
│
├─ Customer dials (inbound)
│   ├─ Want dashboard applets (Voicebot,Passthru, Connect)?
│   │    └─ Exophone → App Engine flow → Voicebot
│   └─ Want logic in your app (no flow)?
│        └─ Exophone → gRPC endpoint → leg actions (ExoML)
│
└─ You dial (outbound)
    ├─ Bot only, fixed StreamUrl?
    │    └─ Connect Voice AI API
    ├─ Need Exotel flow (Voicebot applet/Passthru / agent handoff)?
    │    └─ Connect Voice AI with Flow API
    └─ Need gRPC + leg actions / dynamic stream / parallel say-play?
         └─ ExoML Programmable Legs API
```

Same bot, four front doors

Entry	Who dials	Control	Who configures WSS
Connect Voice AI	Your API	Connect API	StreamUrl in API
Connect + Flow	Your API	App Bazaar	Voicebot/Stream applet
ExoML outbound	Your API (POST /legs)	gRPC + leg actions	start_stream body
Inbound Flow	Customer	App Bazaar	Voicebot/Stream applet
ExoML Inbound	Customer	gRPC + leg actions	start_stream body

References

- Connect Voice AI API
- Connect Voice AI with Flow API
- Programmable Voice APIs with AgentStream
- ExoML – Introduction
- AgentStream developer guide
- Implementing Your gRPC Endpoint for Real-Time Call Events

6.2. Updated Extension Guide: Working with the Stream and Voicebot Applet

Overview

This guide provides a comprehensive overview of how to use Exotel's **Stream Applet** (Unidirectional) and **Voicebot Applet** (Bidirectional) for media streaming applications. These applets enable real-time **audio streaming only** between Exotel and external bot platforms using **WebSocket-based** integration. Note: Exotel does **not** provide inbuilt STT (Speech-to-Text) or TTS (Text-to-Speech) capabilities. You must use your own bot platform to handle media decoding and AI processing.

This article is an **extension** of: Working with the Stream and Voicebot Applet

For stream metadata, logging, and observability, refer to the companion article: Here

Why Choose WebSocket-Based Integration?

WebSockets are ideal for low-latency, bi-directional audio streaming. Compared to SIP or polling mechanisms, WebSocket integration offers:

- **Real-time bidirectional media transfer**
- **Persistent low-latency connection**
- **Lightweight and scalable protocol for AI integrations**
- **Simplified firewall/NAT traversal compared to SIP RTP**

This makes it the preferred method for modern bot platforms such as Dialogflow, Azure Bot Framework, Gupshup, Yellow.ai, Haptik, Google CCAI, Amazon Lex, and in-house NLU/LLM-based bots that support real-time WebSocket-based audio streaming.

Key Technical Concepts from the Core Stream/Voicebot Applet Article

To ensure compatibility with the base setup, this section re-emphasises the following from the core guide:

- **WebSocket Endpoint URL** must be publicly reachable and must support ws:// or wss:// protocol with base64-encoded audio frames.
- **Maximum Connection Time**: Streaming can last up to 60 minutes per session (check limits based on plan).
- **Timeout Handling**: If the bot server does not respond within 10 seconds, the session will fail.
- **Connection Retry**: Exotel will attempt one automatic retry if the WebSocket handshake fails. Ensure redundancy in bot endpoints.
- **Security**: For wss:// endpoints, ensure valid TLS certificates are installed. Self-signed certs may be rejected.
- **Payload Format**: Use base64-decoded Linear PCM payloads (raw/slin 16 bit, 8kHz mono PCM) to feed your STT engines.

These constraints and protocols apply to both Stream and Voicebot applets and should be adhered to during bot deployment.

Applet Differences

Applet	Streaming Direction	Primary Use Case
Stream Applet	Unidirectional	Transcribe user audio, e.g., STT
Voicebot Applet	Bidirectional	Full voice AI interaction (bot speaks + listens)

Note: Exotel only handles the media relay. The bot must provide transcription, NLU, TTS, etc.

Events Sent Over WebSocket

Each applet emits events during the call session. The communication over the WebSocket is bi-directional, with distinct roles for Exotel and the bot:

- **Exotel to Bot:** Transmits session events (Connected, Start, Media, DTMF, Stop, Clear) and the audio stream in base64-encoded chunks.
- **Bot to Exotel:** Returns media (for Voicebot Applet only) and may trigger control markers (e.g., Mark). The bot must gracefully handle session events and terminate the WebSocket connection when the bot conversation ends. Ending the WSS connection triggers Exotel to move to the next applet. There is no explicit Stop event sent from the bot to Exotel.

Each event type Exotel emits over the WebSocket serves a distinct purpose in orchestrating the media stream and enabling seamless bot interaction. Understanding these events allows for optimised bot design, real-time feedback loops, and intelligent call flow transitions.

Best Practices & Use Cases per Event:

- **Connected**: Confirms WebSocket handshake. Use this to initialise your bot pipeline (e.g., STT/TTS service initialization, session state allocation).
  - Best Practice: Log and correlate with call_sid for session tracing.
  - Use Case: Trigger bot intro prompt preparation.
- **Start**: Indicates that audio streaming is beginning.
  - Best Practice: Start buffering/streaming audio to the STT engine.
  - Use Case: Sync with a user-facing prompt like "How can I help you today?"
- **Media**: Continuous chunks of base64-encoded PCM audio.
  - Best Practice: Ensure your STT engine handles 100ms PCM blocks efficiently.
  - Use Case: Real-time speech transcription or voice intent detection.
- **DTMF**: Keypress detection on the caller's side.
  - Best Practice: Use to branch hybrid IVR+Bot logic without STT latency.
  - Use Case: Press 1 to speak to agent → Exotel hears DTMF → triggers escalation.
- **Mark**: Bot-sent event to indicate a logical milestone.
  - Best Practice: Use to sync analytics or inject debugging hooks.
  - Use Case: Mark when the bot completes a sub-flow (e.g., "address collected").
- **Stop**: Triggered by Exotel when the customer's leg is disconnected.
  - Use Case: To identify when the customer's leg has disconnected.
- **Clear**: Resets session context mid-call. Sent by Voicebot. Voicebot sends this event to indicate that the current conversational context should be flushed and re-initialized. This is useful in scenarios where the bot needs to start a fresh session mid-call, such

as when the caller says “start over” or the previous context is corrupted.

- Best Practice: Ensure your bot wipes session memory and re-establishes a clean state when a Clear event is received.
- Use Case: Caller says "start again" → Clear received → bot resets all entities, replays welcome prompt.
- Supported Chunk Size: Ensure the bot handles minimum 100ms audio chunks (approx. 3.2 KB base64 payload per frame) for seamless reset and session realignment during mid-call context switches.
- Best Practice: Use to reinitialise bot memory (e.g., context drops).
- Use Case: Caller says "start over" → bot requests Clear → reset the conversation.

These events should be monitored in real time and mapped to your backend decision logic and conversation orchestration flow.

1. Connected: Indicates the WebSocket connection is successfully established. (Initialize bot session, allocate STT/TTS/LLM resources)
2. Start: Voice media stream is starting. (Start decoding audio for transcription or detection)
3. Media: Base64-encoded PCM audio payload. (Feed to STT or analytics pipeline)
4. DTMF: DTMF tones detected. (Capture digit input in IVR scenarios)
5. Mark: Developer-defined markers. (Sync checkpoints in bot logic)
6. Stop: Media streaming session ends. (Escalate, save transcript, or trigger routing)
7. Clear: Reset session context. (Re-initiate bot logic mid-call)

Streaming Audio Format

- **Codec**: 16-bit Linear PCM (s16le)
- **Sample Rate**: 8000/16000/24000 Hz
- **Channels**: Mono
- **Encoding**: base64 in WebSocket frames

Dynamic URL and Custom Parameters

You can configure a dynamic WebSocket URL using placeholders and custom parameters.

Custom Parameter Rules:

- Max 3 parameters
- Total param length (after ?) must not exceed 256 characters
- Sample:

ws://127.0.0.1:5001/media?param1=value1&param2=value2&param3=value3

- Dynamic resolution from HTTP(s) applet must also return a valid ws(s) URL in this format.

Recording (Optional)

Recordings, if enabled, are returned via Passthru as a RecordingUrl.

Use Cases:

- Train STT/LLM models

- Compliance and QA reviews
- Escalation audits
- Voice sentiment labeling datasets

Routing to Agent or Contact Center After Voicebot Applet

When the **WebSocket connection is closed**—either due to the bot disconnecting after completing its interaction or a network-level termination—Exotel automatically moves to the next applet in the flow. There is **no explicit Stop event sent by the bot to Exotel**. Instead, **the bot must close the WebSocket session** once the conversation ends.

Upon disconnection, Exotel internally emits a Stop event and transitions to the next applet, typically a **Passthru Applet**. This Passthru makes a GET request to your endpoint, and based on your response (e.g., escalate=true), Exotel decides how to route the call.

Scenarios:

- **Connect Applet** → Route to Exotel agent
- **SIP Connect via vSIP Trunk** → Route to enterprise contact center
- **Hangup Applet** → Gracefully end the call

Example:

User: "Talk to human" → Bot finishes → WSS disconnects → Exotel emits Stop → Passthru GET call → Response: escalate=true → SwitchCase → vSIP Trunk over Connect applet

Passthru Integration

Always place a Passthru Applet immediately after Voicebot/Stream Applet to:

- Fetch session metadata
- Log streaming stats (SID, duration, Recording URL)
- Detect disconnects
- Read escalation flags from response

Best Practices

- Place Passthru immediately after streaming applet
- Use Clear/Mark events for context handling and observability
- Use Active Streams API for concurrency limits
- Design Passthru logic to interpret escalation/disconnect
- Follow WebSocket timeout, reconnect, and handshake guidelines strictly
- Keep custom params concise and secure
- Ensure bot sends Stop to gracefully close stream

Advanced Use Cases

- STT pipeline using OpenAI Whisper, Google STT
- Bot+LLM conversations via voice
- IVR replacement with NLP flows

- DTMF + Speech combo journeys
- Hybrid fallback to live agents via SIP trunk

Summary

Exotel's Voicebot and Stream Applets power modern voice automation by offering developer-first WebSocket-based audio streaming infrastructure. These applets act as programmable media bridges, streaming audio in real time between Exotel's telephony platform and any compliant bot platform capable of understanding linear PCM data.

By adopting this architecture, enterprises gain:

- **Low-latency streaming** that supports responsive AI-driven conversations
- **Vendor-neutral integration** with STT, TTS, LLMs, and custom NLP systems
- **Dynamic routing and escalation** via Passthru Applets and intelligent fallback logic
- **Secure, observable, and resilient media sessions** with active stream monitoring
- **Enterprise-grade call flow design** that integrates Exotel CC, SIP trunks, and external agents

This document is a production-ready extension to the Stream/Voicebot Applet Basic Guide and complements the Passthru Streaming Metadata Guide.

For deployment, ensure compliance with session lifecycle best practices, chunk size handling (100ms), parameter limits, recording strategies, and escalation routing logic through WebSocket termination events.

6.2.1. More

Capabilities Overview

- Real-time audio streaming over WebSocket (WSS)
- Voicebot Applet: Bidirectional audio interaction
- Stream Applet: Unidirectional audio streaming
- Passthru Applet: Stream lifecycle metadata, error reporting, and routing context
- SIP Trunk fallback and external call routing available via Passthru (Mumbai region only)
- Supports major bot platforms.

Best Practices

- Always choose the Mumbai instance for IP–PSTN or SIP hybrid integrations.
- Disconnect the WebSocket from your bot when the conversation is complete to allow the call flow to proceed to the next applet.
- Ensure media chunking follows Exotel's recommended size (typically 100ms PCM chunks with 3200 bytes of raw audio).
- Use the Passthru Applet after Voicebot to capture stream completion, errors, and call routing metadata.
- Avoid long delays in establishing WebSocket connections to prevent streaming failure or fallback.
- Use dynamic WebSocket URLs when handling multi-tenant or user-segmented bot sessions.
- Monitor StreamSID, Duration, Status, and DisconnectedBy via Passthru for complete lifecycle debugging.

Glossary

- **WSS (WebSocket Secure)**: Secure protocol used for encrypted audio streaming.
- **Voicebot Applet**: Enables bidirectional audio exchange between the bot and the caller.
- **Stream Applet**: Allows unidirectional audio capture from the caller to your bot.
- **Passthru Applet**: Captures stream metadata, errors, and acts as a decision trigger for routing.
- **StreamSID**: Unique identifier for each stream session, useful for logging and tracking.

Common Use Cases

- Voicebot handling FAQ or collections for fintech and NBFCs
- Call deflection bots for first-level support in marketplaces
- Healthcare appointment management via AI bots
- Bidirectional streaming for GenAI-based interview bots
- Smart routing from bot to agent based on sentiment or keywords

Once these steps are completed and access is approved, you will be ready to stream calls to your own bot infrastructure in real time and build powerful conversational workflows using Exotel's infrastructure.

For questions, assistance, or deployment readiness, contact your CSM or raise a support request via hello@exotel.com.