Twilio integration with Cisco Unified Contact Center Enterprise

June 2025

Document History

Revision No.	Date	Description	Author
1.0	June-20-2025	Configuration Guide	Simal Mathai

Table of Contents

Table of Figures

1 Introduction

This document is intended for technical staff and Value Added Resellers (VAR) with installation and operational responsibilities. This document provides the configurations required for the integration of Twilio Studio, a platform for PSTN connectivity plus tools for building IVRs and chatbots (including integrations with Generative AI-capable Virtual Agents, like Dialogflow CX integrated with Google), with Cisco Unified Contact Center Enterprise (UCCE). As a result of this integration, a UCCE Agent will receive calls with call context details supplied from the Twilio Studio IVR / Virtual Agent, regarding its prior processing of the call, when answering incoming calls from the PSTN. Configurations related to integration of UCCE components are not covered in this document.

1.1 Executive Summary: Example Use Case Scenario, Benefits

This Configuration Guide "validated blueprint," while modeled after a specific customer use case, is generic enough that it can represent almost any attempt to front-end a customer's Cisco UCCE contact center with a developer-built, customized/bespoke, modern, cloud-based IVR or Virtual Agent/bot from Twilio, where the sending along of spoken, entered, or queried data – i.e. "Call Context" – to Cisco contact center Agents (e.g. as a screen pop in their Finesse agent desktop application) is needed.

The Twilio IVR in this scenario can be a basic DTMF- or speech recognition-based IVR (Interactive Voice Response system), or a fully conversational IVR built with an AI-based virtual agent (such as Google's DialogFlow CX on Twilio), as the customer may prefer to build. The job of that IVR or Virtual Agent – successfully validated by this blueprint and TekVizion's testing of it – is to supplement and/or eventually replace the existing IVR built into a customer's Cisco UCCE system (CVP), for purposes of providing bespoke, intelligent AI-based automated self-service via a Virtual Agent "bot," while still providing the necessary call context for any calls being redirected to the Cisco UCCE contact center, so that human agents there can still process the remaining incoming calls being escalated to live agents, as needed, without callers having to repeat themselves to give the human agents the same (call context) info they already gave the Virtual Agent / bot.

In this particular case (see Figure 1 below), we've modeled our scenario on that of a typical retail customer who contacted Twilio for help, because they had a workflow that – while mostly automate-able – they noticed also needed to be able to handle escalations on few corner cases of orders to a human agent (connected via UCCE), seamlessly. Other use case examples where this has proven helpful (but not documented here, being similar but more complex) include another retailer who had a complex collection of service departments providing customers with help, depending on what kind of help/service they needed and what of the companies' products they owned, etc. Companies that have to route incoming calls from their customers to the appropriate facilities/locations able to service those customers, provide customers with self-service information relevant to their purchased modes/products, and/ or help them set up service appointments for the appropriate service tasks with personnel in those locations – without the calling customer having to repeat their information or what they were looking for/what they wanted multiple times – can all use similar set ups to this, without having to forklift replace their Cisco and UCCE/CVP Contact Center tech estate.

In the case of the particular customer we've modeled this tested/blueprint scenario after, their existing Cisco Contact Center and IVR system also had several limitations leaving it unable to address many self-service use cases by itself, such as:

- The inability to also send text messages (with helpful self-service or TFA - Two Factor Authentication - links),
- An inability (at least not without expensive firmware upgrades and software upgrade service contract renewals) to add speech rec or AI-based conversational Virtual Agent capabilities,
- The practical inability for the IVR/AI-based Virtual Agent to be shared across multiple sites / Cisco on-premises installations from the cloud, difficulty in making changes and or to be centrally managed from the cloud.

All of which led this customer to want to front-end their Cisco Contact Center system with something more modern, which they could customize as needed, and build for themselves in the cloud, on Twilio, servicing all of their locations and enabling them to provide a higher level of (personalized) service as well as a higher degree of self-service to their calling customers.

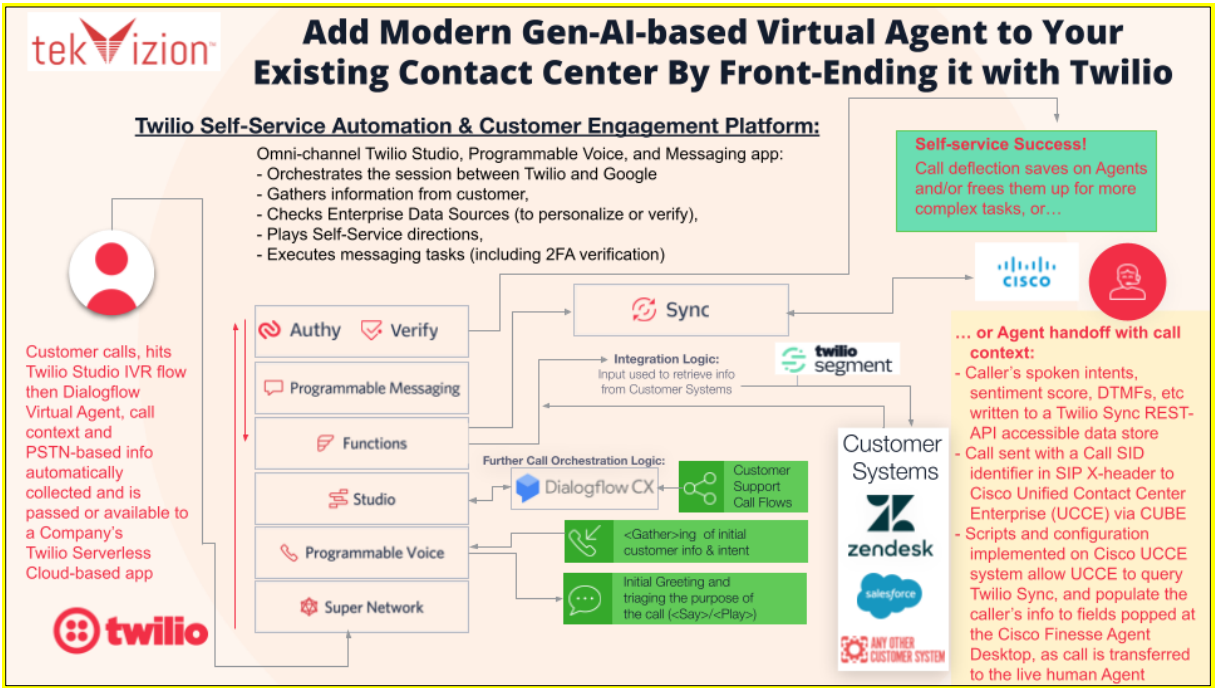

Figure 1 A flowchart view of the use case tested in this TekVizion Validated Blueprint

When we therefore began constructing our Configuration Guide with this use case in mind, we were seeking to specifically address a typical call flow similar to that faced by a caller into this company's services number, as follows:

- The caller first calls into a Twilio IVR / Virtual Agent (same phone number, owned by or ported to Twilio, of the company's customer service numbers) similar to the one created here, and is greeted and prompted for various order data (and/or have that data looked up by the Twilio IVR/Virtual Agent), such as stating that they wanted to order a certain kind of product, with certain add-ons (which might or might not be available in a particular location), etc.

- Then the caller would then be potentially directed to some self-service ordering and fulfillment questions in the Virtual Agent, with answers delivered to the customer via played prompts and/or text messages sent, for getting the product and service they required.
- If that addressed the caller's need, in a self-service/automated manner, then… success! (as in many cases)!
- Else… in cases where the self-service information was not enough, or the caller still needed service (e.g. a particular requested add-on product, say "Anchovies" on a pizza order, that was not available at a particular location and a substitute was needed, or an order change required), then the caller could elect to have their call sent (queued) at that point to a live human contact center Agent staffer, who could help the caller with those tricky details – but most importantly, do so *with all of the call context already input and looked up by the Twilio platform and Dialogflow CX Virtual Agent up to that point* sent along to that human agent also, including:

    - DMTF keypresses, called number and whatever number the caller had called from (ANI/DNIS), language preference entered, etc. (the call metadata),
    - what products previously ordered (as well as current order) or models the customer owned and what they were calling for service about (i.e. the relevant media content of the call)
    - interaction history with the bot (last matched intent, and detected caller sentiment [score] at last matched intent), etc.

    - So that the human agent in the Cisco UCCE Contact Center eventually picking up the call would have all of that call context already in front of them upon answering the call in their Finesse Agent desktop application, without the need for the caller to repeat any of this info, or for the agent to take time to re-gather this information. Instead, the agent can "get right to the point" with the caller.

Note that while not every literal potential step in the above flow alluded to above is covered here in this Configuration Guide (it would be too long), the rest can predominantly be found among Twilio's Quickstart guides online. The link for Twilio's Quickstart guides online is https://www.twilio.com/docs/voice .

However, the basic framework and necessary steps for how to collect and successfully pass entered call context from a customer-built Twilio app to an Cisco on-premise Unified Contact Center Enterprise system are laid out here, step by step.

The main tools used to build this validated call flow include Twilio Studio, and the Twilio Console interface to other Twilio services and tools (such as Functions and Sync), Twilio's one-click integration with Google Dialogflow CX, as well as the Cisco UCCE's various admin interfaces, as described below.

One interesting key wrinkle solved in validating this Configuration Guide was that the Cisco UCCE contact center, as is typical for an on-premises contact center system, naturally assumes the existence of an on-premise database from which info like call context would be assembled to be sent on to agents (instead of all the info arriving with the call itself from off-board) – logical enough in the "old days" of single-vendor vertically integrated on-premise systems, but limiting the Cisco system's usefulness and extensibility today – whereas more modern, multi-party cloud-based

bespoke and/or customer-built solutions typically use data sources also centralized in the cloud for sending data with arriving calls as they are passed around. Read on below to see how ancillary Twilio tools like Twilio Sync have enabled the key "cloud database-to-premises system connectivity" aspect necessary to get this configuration working for Cisco customers in those cases!

1.2 TekVizion Labs

TekVizion Labs™ is the first independent interoperability certification lab for business voice communication leaders. TekVizion's future-proofed Labs is capable of on-demand testing to virtually any network configuration and need. Other benefits include:

- Gain a competitive edge by accelerating integration to hundreds of collaboration solutions and ensure end-to-end functionality for any of your customer's unique network requirements.
- Virtualized instances of 300+ collaboration products to build and teardown complex configurations on demand.
- Launch new features with agility and maintain quality and reliability when validating end point devices.

About TekVizion

TekVizion is the unparalleled innovator in validation and automation for collaboration solutions. Our unique combination of systems, automation, and expertise are optimized for quality, reliability, and scalability for businesses who build or offer collaboration solutions. We enable our customers to release new features, automate tasks, and deliver proven communications that help grow revenue in a more efficient, cost effective, and timely manner.

As the most experienced independent vendor, TekVizion has partnered with global service providers and others as our valued customers. Contact us at www.tekvizion.com for more details.

1.3 Twilio Programmable Voice: <Gather>, Studio, Dialogflow Virtual Agent integration, Functions and Sync, etc.

Twilio's Programmable Voice powerful suite of Communications Platform-as-a-Service (CPaaS) of tools to control and manipulate calls (programmatically) placed to Twilio owned or ported Phone Numbers includes the following:

Twilio <Gather> speech recognition, for the latest and greatest in speech input collection from any of multiple providers, with our patent-pending speech provider failover technology, with the most advanced speech models and a wide variety of languages (including multi- language detection capability and support). More about the latest on Twilio's speech recognition technology is available from below links:

https://www.twilio.com/en-us/changelog/-gather--new-multi-provider-speech-recognition-models---upcoming

https://www.twilio.com/docs/voice/twiml/gather#overview

Twilio Studio is a visual, drag-and-drop editor for creating applications. It can help to build an entire IVR system itself, or to orchestrate the flow of calls between components such as a Dialogflow Virtual Agent from a bot provider like Google, with easy one-click connectors, as well as other Twilio components like PCI-compliant **Twilio <Pay>** for taking credit card payments over the phone . When using Twilio Studio, customers will also often use Twilio Functions, Assets, Sync, and other aspects of Twilio's platform in building their applications to connect the piece parts programmatically, to do things like pass call context amongst them.

Read more about them here:

Twilio Studio: https://www.twilio.com/en-us/serverless/studio
Twilio Functions: https://www.twilio.com/en-us/serverless/functions
Twilio Sync: https://www.twilio.com/docs/sync/api

Additionally, Twilio Studio comes with many "one-click" built-in widgets that make adding in partner capabilities, such as linking to **Google DialogFlow CX** AI-based Virtual Agents for Conversational IVRs, much easier. Refer below link for more on how easy to integrate and use it.

https://developers.twilio.com/Twilio-Developers/IVR-Build-a-thon?exit_from_webinar=true&auto_enter=cancelled&t=1

1.4 Cisco Unified Contact Center Enterprise

Cisco Unified Contact Center Enterprise is a solution that delivers intelligent call routing, network-to-desktop Computer Telephony Integration (CTI), and multichannel contact management to contact center agents over an IP network. Cisco UCCE combines software IP Automatic Call Distribution (ACD) functionality with Cisco Unified Communications to enable companies to deploy an advanced, distributed contact center infrastructure rapidly.

The Unified CCE product integrates with Cisco Unified Communications Manager, Cisco Unified Customer Voice Portal, Cisco VoIP Gateways, and Cisco Unified IP Phones. Together these products provide contact center solutions to achieve intelligent call routing, multichannel ACD functionality, Voice Response Unit (VRU) functionality, network call queuing, and consolidated enterprise-wide reporting. Unified CCE can optionally integrate with Cisco Unified Intelligent Contact Manager to network with legacy ACD systems while providing a smooth migration path to a converged communications platform.

The Unified CCE solution consists primarily of four Cisco software products:

- Unified Communications infrastructure—Cisco Unified Communications Manager
- Queuing and self-service—Cisco Unified Customer Voice Portal (Unified CVP)
- Contact center routing and agent management—Unified CCE. The major components are CallRouter, Logger, Peripheral Gateway, and the Administration & Data Server
- Agent desktop software—Cisco Finesse

2 Topology

The network topology used for Twilio Call Studio with Cisco UCCE integration is shown below. A registration SIP trunk on UDP is configured between Twilio and UCCE Ingress Gateway, Cisco UBE.

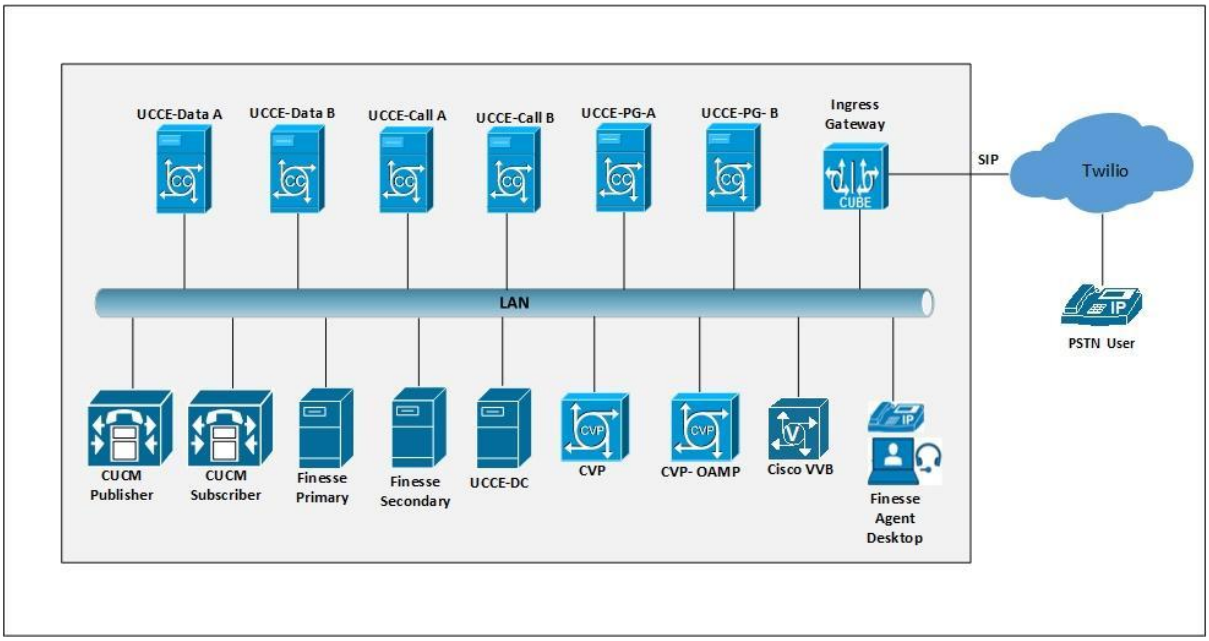

Figure 2 Network Topology

2.1 UCCE Hardware Components

Hardware Device	Version
VMware ESXi Server (Running UCCE Server Virtual Machines)	7.0.3
Cisco Unified Border Element - ISR4331/K9	14.7

2.2 UCCE Software Requirements

Software Components	Version
Cisco Unified CCE	12.6(2)
Cisco CVP	12.6(2)
Cisco VVB	12.6.2.10000-25
Cisco Finesse	12.6.2.10000-82
Cisco UBE	17.12.4a
Cisco UCM	15.0.1.12900-234
Microsoft DNS	1.0

3 Pre-requisites

3.1 Twilio

- Twilio account
- Twilio phone number
- Twilio Studio to build IVR flow
- Google GCP account, in which to build a Dialogflow CX Virtual Agent, if desired

3.2 Cisco UCCE

- Cisco UBE to have a public interface configured which is required to setup the Trunk to Twilio
- CVP is configured to send a new call request to Cisco ICM via the Peripheral Gateway.
- Unified CCE is integrated with CVP, VVB, Finesse and Cisco UCM.
- ICM having the Dialed Number and Call Type configured which is required for the mapping of ICM script.
- Cisco Finesse Agents configured to accept the incoming calls to UCCE.

4 Detailed Twilio - Cisco UCCE Call Flow

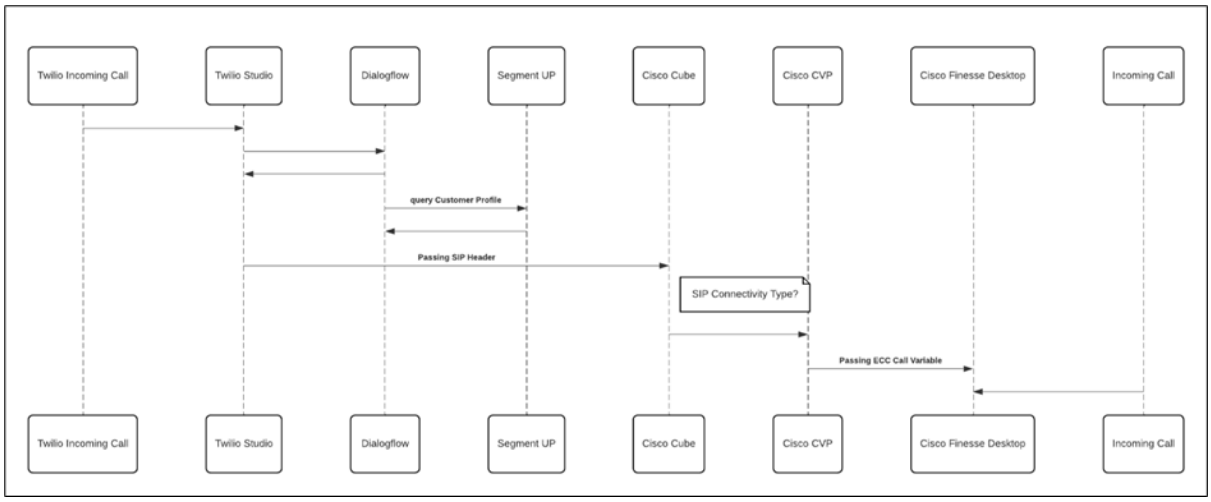

Figure 3 Call Flow

- PSTN user calls a Twilio-owned number procured, and/or ported to Twilio, by the operating organization
- The Twilio Platform routes the call to that organization's IVR, created in a Twilio Studio Flow, including prompting the user with a <Gather> for some initial info or for an initial intent, and from there integration to a Dialogflow Virtual Agent, and, potentially, a Customer Profile lookup in a Unified Profile before connecting the call to that bot/IVR.
- The Call context of the caller's conversation with the bot / IVR is stored, via Twilio Functions and Sync, in a format and data repository such that the Cisco environment will later be able to access this information when passing the call to the answering Agent
- Based on the caller's selections in the IVR/Virtual Agent the call can be escalated ("Escalate to Live Agent in the Studio flow" – see diagram below) i.e. sent to the Cisco environment, where it will be routed to the appropriate Agent queue in the Cisco UCCE.

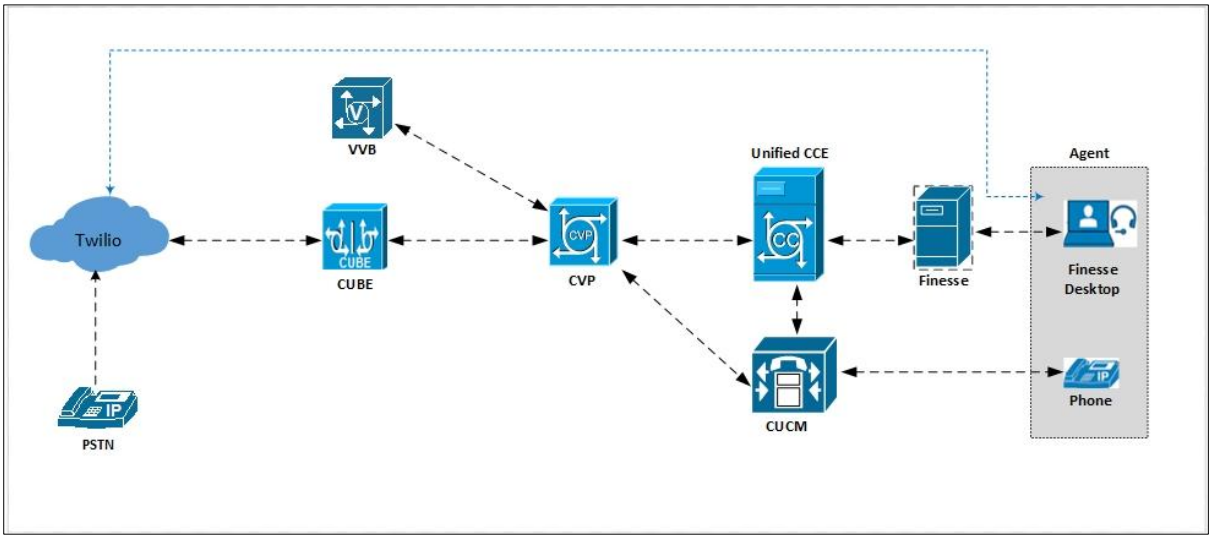

Figure 4 Call Flow Cisco UCCE side

- Once routed to Cisco UCCE, Cisco UBE receives the call INVITE from Twilio with a header called "x-ciscodata" which contains the Twilio call ID
- Cisco UBE sends the call to Unified CVP, which is configured to pass the "x-ciscodata" header to the Unified CCE.
- Unified CVP sends a new call request, which invokes a new incoming Dialed Number, which further invokes a routing script in the Unified ICM.
- The Unified ICM routing script determines the need of transferring the call to VVB.
- ICM processes the various defined nodes in the routing script and continues the communication between ICM, CVP and VVB until an Agent becomes available.
- When the Agent becomes available Unified ICM dequeues the call and sends a disconnect to VVB. ICM then sends a connect- to- agent request to CVP.
- CVP then passes this request to CUCM via a SIP INVITE and the call is routed to the Agent.
- The Twilio call ID passed in the "x-ciscodata" header of the Invite is extracted during the execution of ICM routing script, and is saved, as a kind of "key," to a Peripheral Variable which is then passed to the Finesse Agent.
- Finesse Agent sends an API request to Twilio using that key, the Twilio call ID received in the x-ciscodata" header, and displays the call context in the Finesse Agent Desktop as a Screen Pop as the call arrives to the agent.

5 Configuration

This section includes the configurations required in Twilio and Cisco UCCE components for providing the Twilio call context details at Cisco Finesse Agent.

5.1 Twilio Configuration

5.1.1 SIP domains

- Navigate to **Voice > Manage > SIP domains**
- Select the Plus icon to add a new Domain and Under **Configure** provide:
    - FRIENDLY NAME: **Cisco UCCE Demo**
- SIP URI: **ciscouccedemo**
- Under **Voice Authentication**,
    - IP ACCESS CONTROL LISTS: Select the IP access control lists to authenticate inbound calls to Twilio (Refer Section 5.1.2)
    - CREDENTIAL LISTS: Select the appropriate Credential list for authentication (Refer Section 5.1.3)

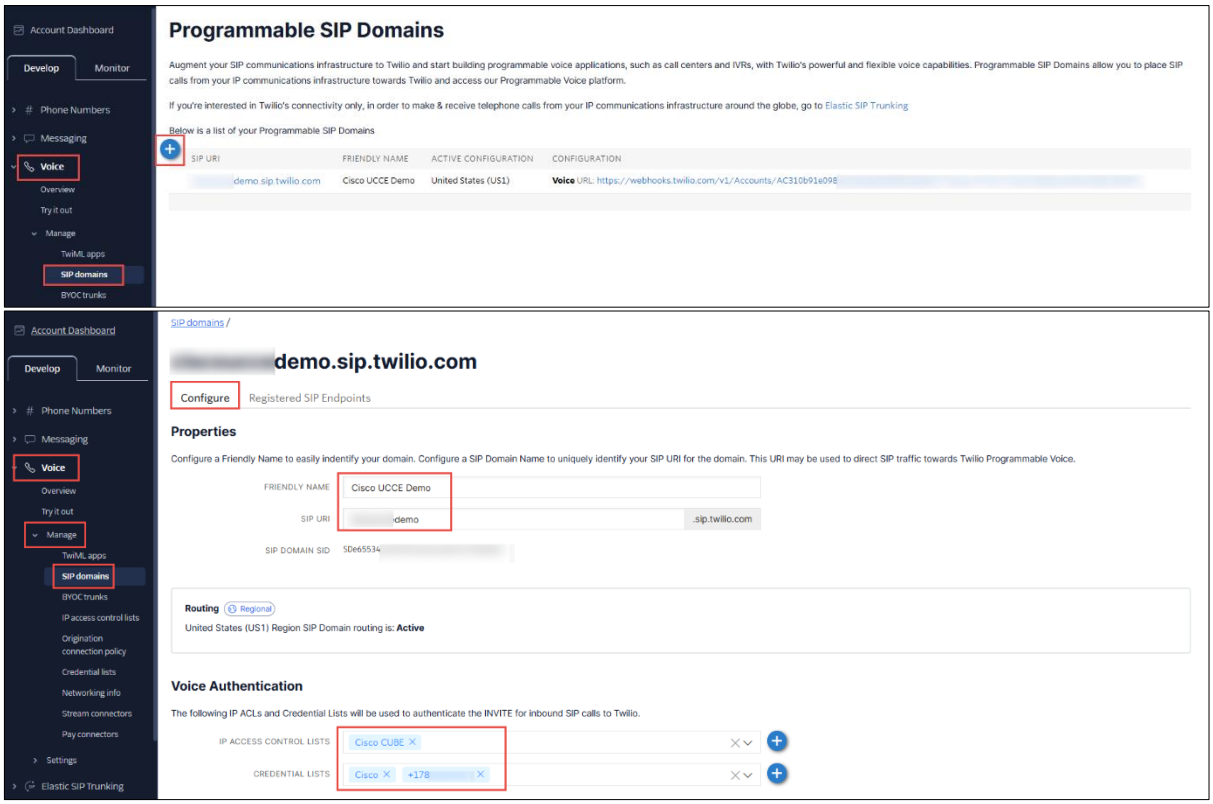

Figure 5 SIP Domains

- Configure the **Call Control Configuration** as follows:

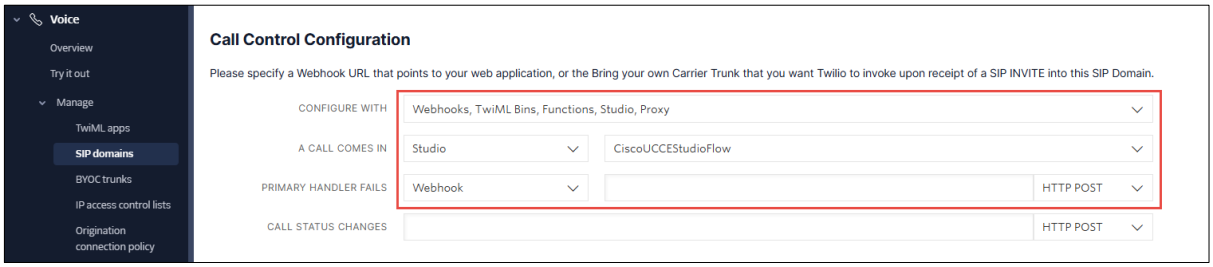

Figure 6 SIP domains continuation

- Under SIP Registration,
  - Allow SIP Endpoints to register: **ENABLED**
- Under **SIP Registration Authentication**,
  - CREDENTIAL LISTS: Select the appropriate Credential list for authentication of SIP Endpoint (Refer Section 5.1.3)

- Save the configuration

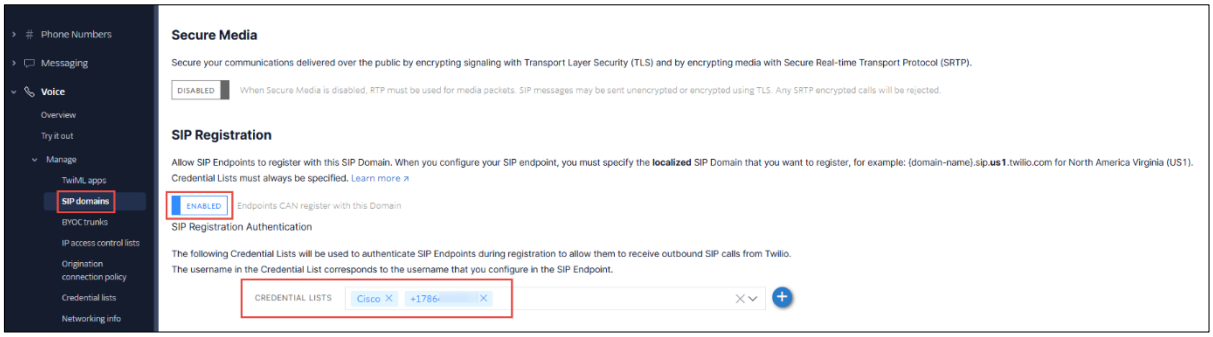

Figure 7 SIP domains continuation

5.1.2 IP access control lists

- Navigate to **Voice > Manage > IP access control lists**
- Click **Create new IP Address Range**
- Friendly Name: **Cisco CUBE**
- IP Address Range: Enter the CUBE WAN interface IP Range
- Friendly Name: Enter a suitable name

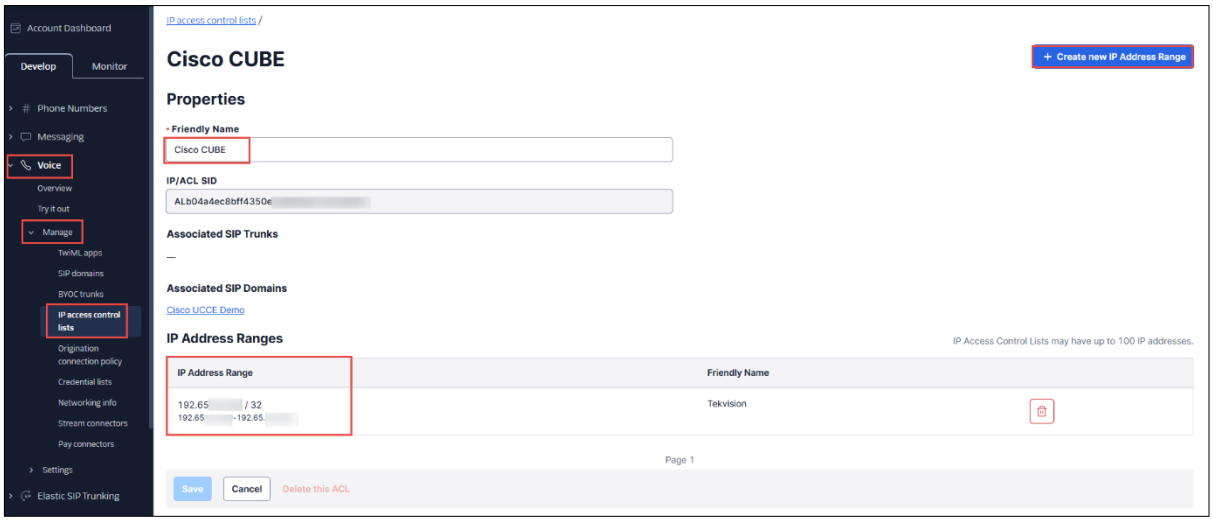

Figure 8 IP access control lists

5.1.3 Credential lists

- Navigate to **Voice > Manage > Credential lists**
- Click the plus sign to create new Credential List
- Friendly Name: Enter a Friendly name
- Username: Enter the Username
- Password: Enter the Password

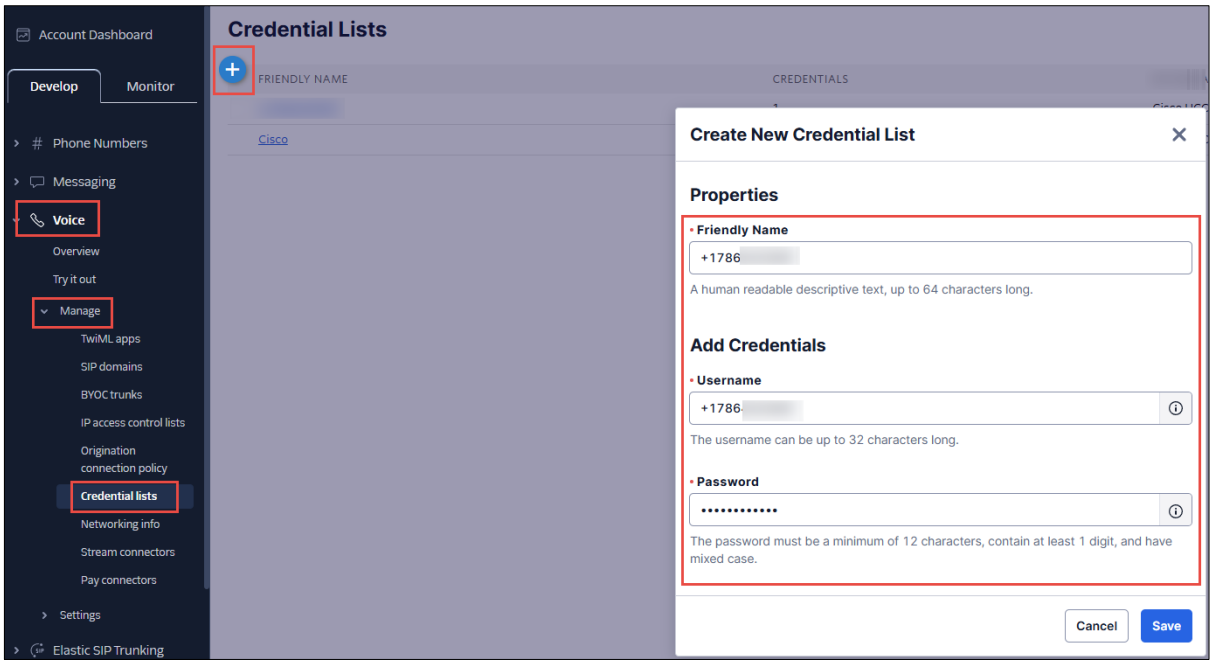

Figure 9 Credential lists

5.1.4 Twilio Studio IVR Flow & Dialogflow Virtual Agent

Twilio IVR Studio flow is created to connect to a Virtual Agent / IVR and route the call to back to Cisco UCCE based on the input provided by the caller. Twilio sends the Call ID to Cisco UCCE in the X-Header of the SIP INVITE.

- Follow the steps in the Twilio Virtual Agent Dialoglfow Onboarding Guide to integrate your Twilio Account and the Google GCP (Google Cloud Platform) account in which you want to build your Dialogflow Virtual Agent. That will create a Studio Flow integrated with the Dialogflow Virtual Agent
- If that does not work, or if you need to edit the Studio Flow connecting to the Dialogflow Virtual Agent later, you can use the Explore Products option in the left pane, navigate to **Studio > Flows** to edit the flow.

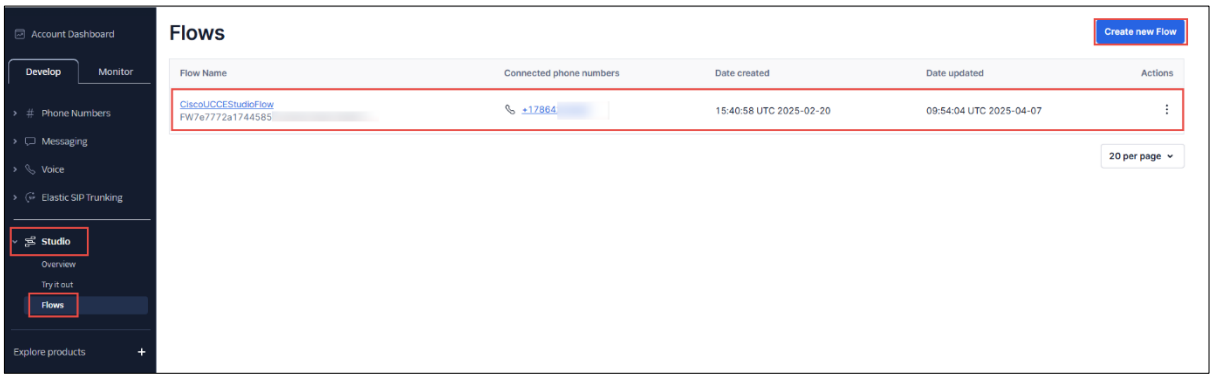

Figure 10 Twilio Studio IVR Flow

- Twilio call flow designed for Cisco UCCE is shown below

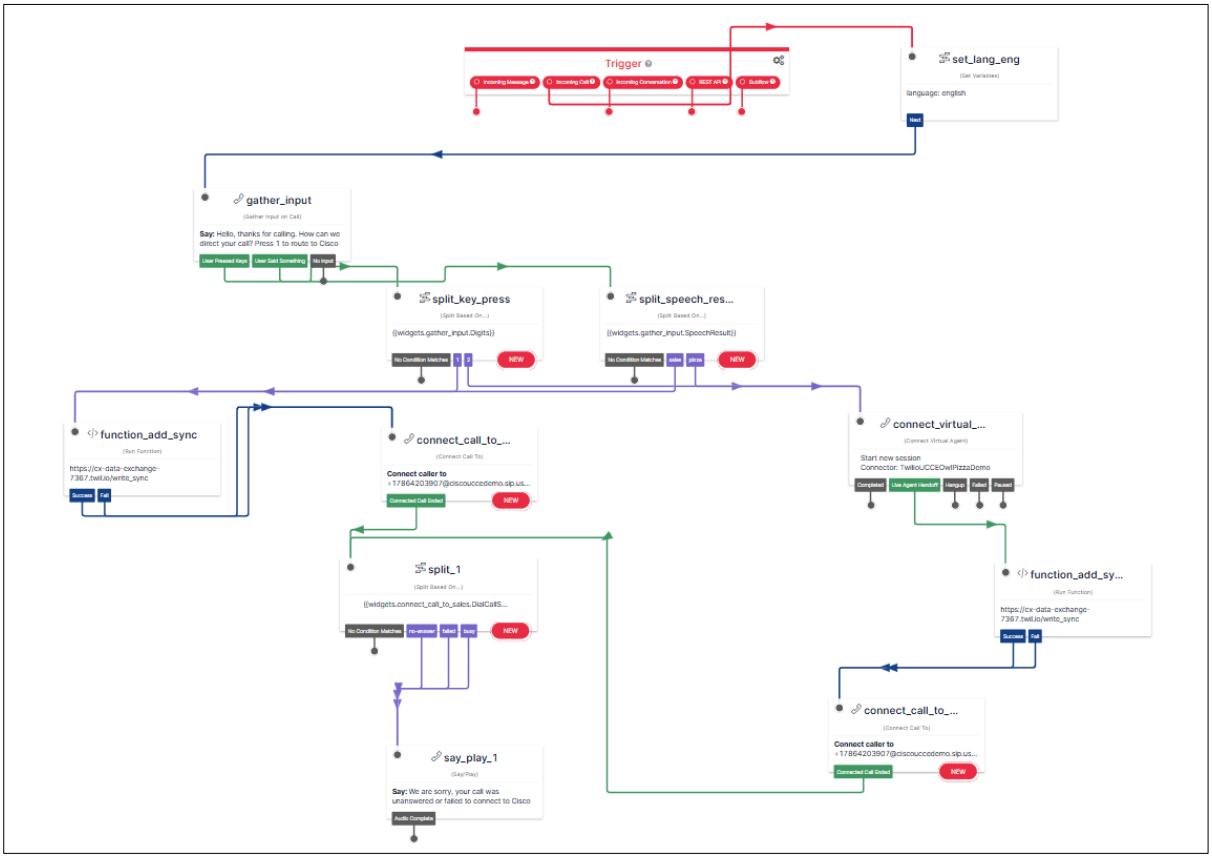

Figure 11 Twilio IVR Studio Flow – Template

General Overview of Twilio Studio Flow for UCCE Front-ending.

As is shown in Figure 11, the six major steps for the call are as follows:

1. The call comes into a specified PSTN number associated with a specific Twilio account, <Trigger>ing there the execution of a Twilio Studio Flow, as configured,
2. whereupon some initial input is collected (Language, and callers initial intent) via a <Gather> (a logical flow chart box, or "widget", in that Studio flow),
3. and then, based on spoken input gathered thereby, the call is directed (via <Split> Studio widget) to…
4. a Dialogflow Virtual Agent (via <Connect_to_VirtualAgent> Studio widget),
5. and when (as an exit state/condition detected) Escalation to a Live Agent is called for in the Dialogflow bot, a <Function_add_sync_virtual_agent> widget is triggered to pass along preserved call context,
6. as the <Connect_call_to> widget is used to deliver the call ultimately to UCCE for routing to an agent/skill group.

Other <Play>/<Say> widgets and <Split> key press widgets are used for trouble shooting and error messaging in the flow.

Detailed view of Twilio Studio flow for UCCE Integration

Widget: Trigger

- Studio Flows can be triggered by an Incoming Message, Incoming Call, or REST API request. Attach Widgets to the trigger events that the Flow should respond to.

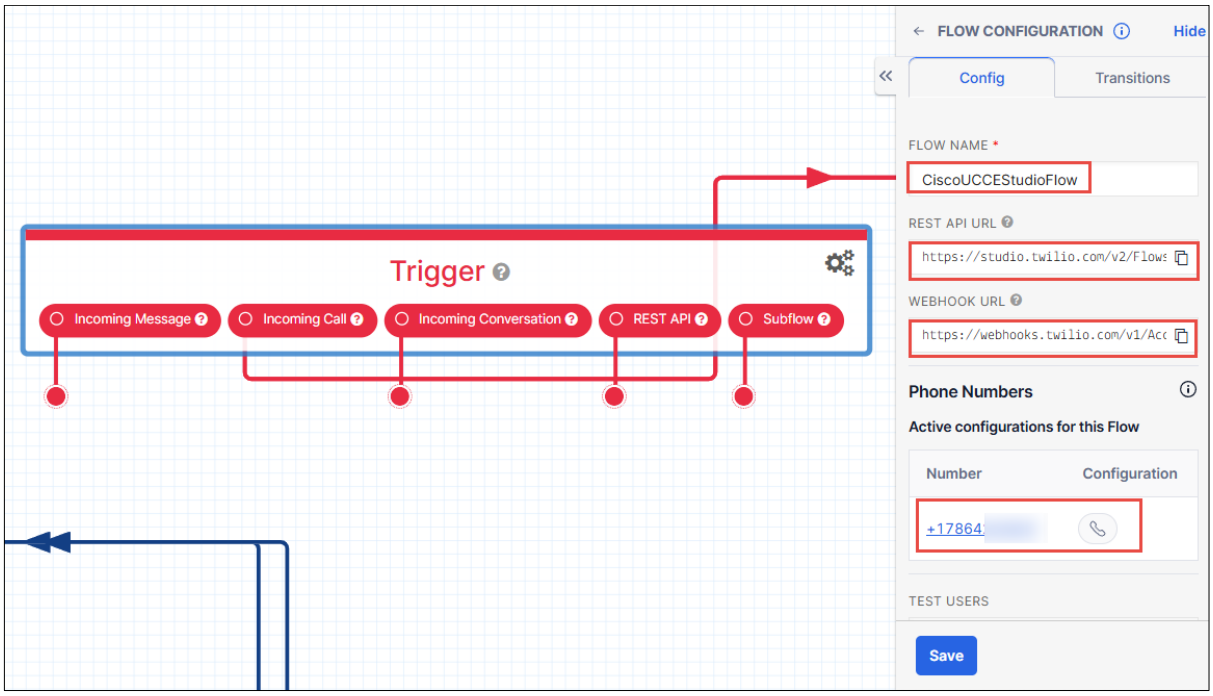

Figure 12 Twilio IVR Studio Flow – Trigger

Widget: set_lang_eng

- Using this widget the language is set to English

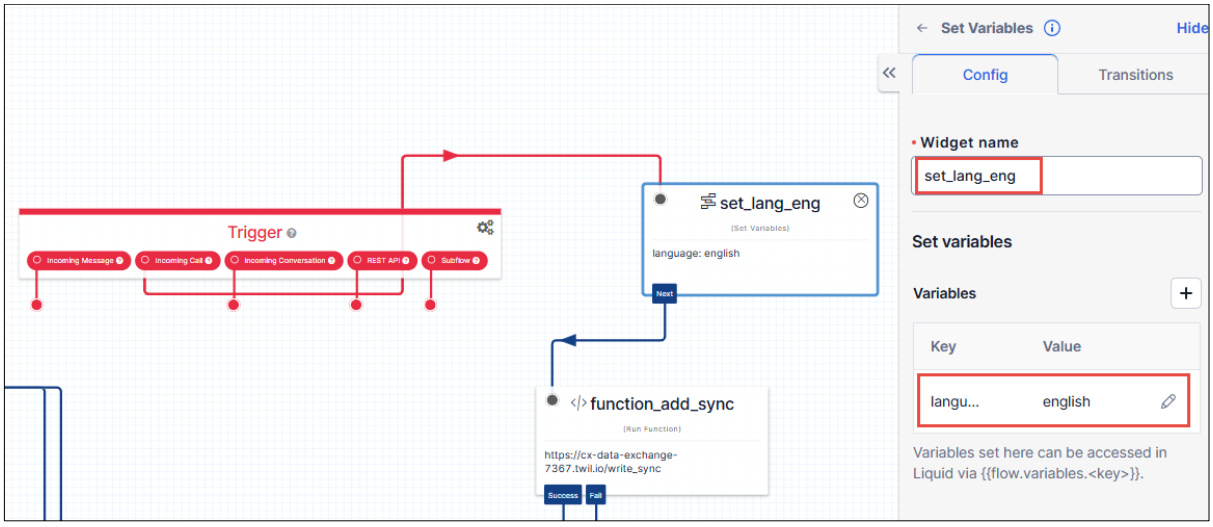

Figure 13 Twilio IVR Studio Flow – set_lang_eng

Widget: gather_input

- This widget gather_input with initial prompt

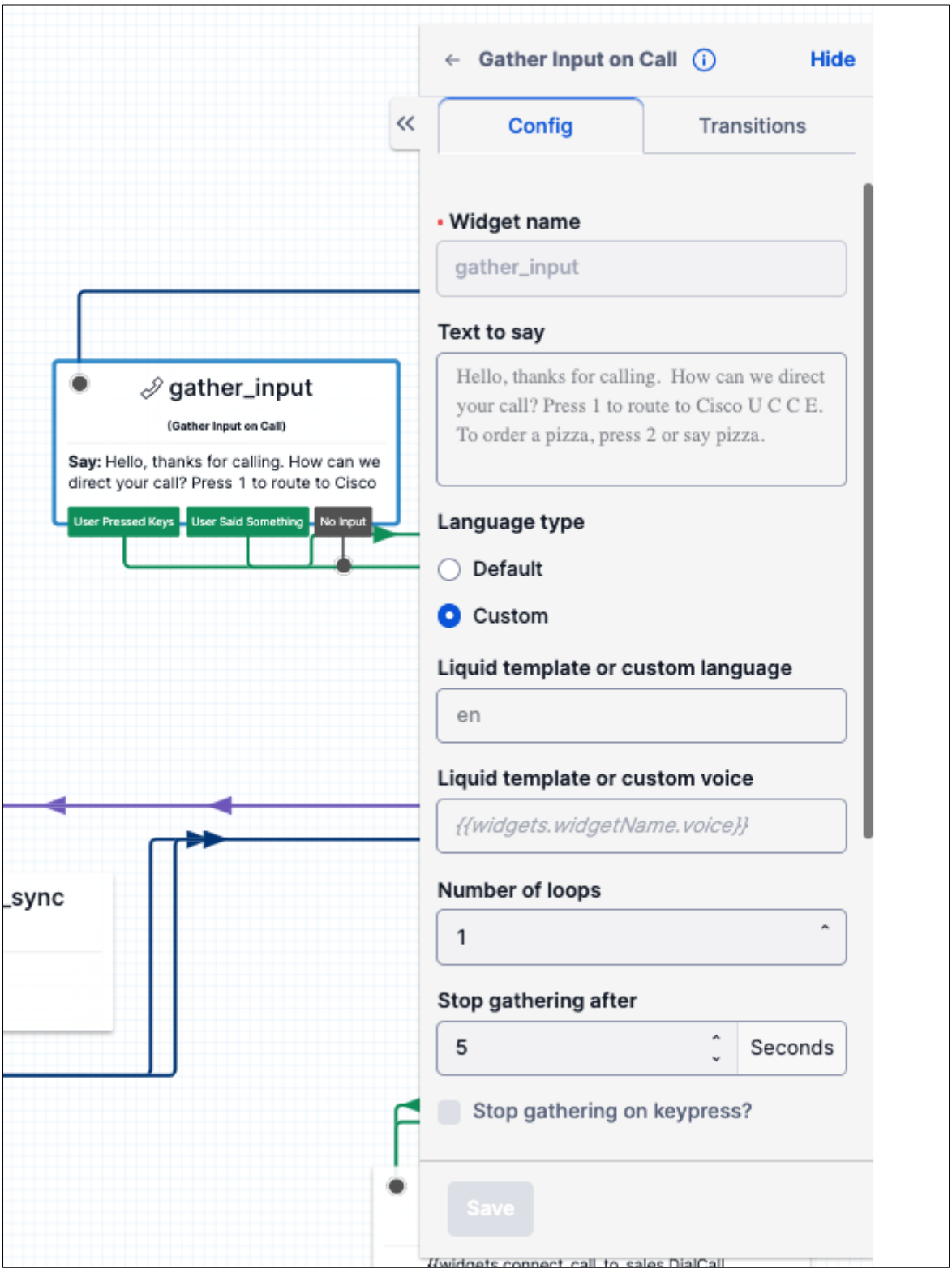

Figure 14 Twilio IVR Studio Flow gather_input

- This widget gather's Advanced speech recognition settings

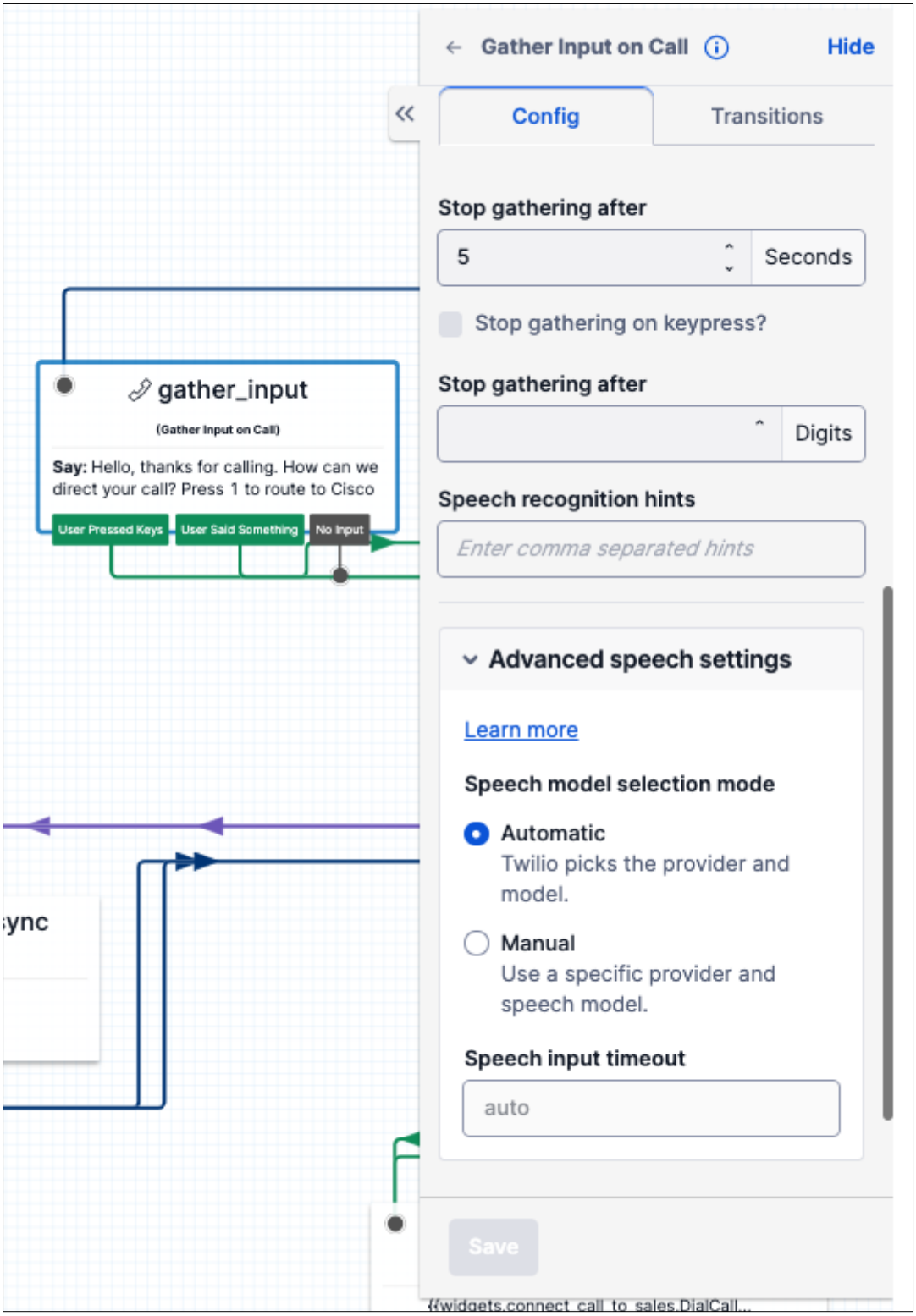

Figure 15 Twilio IVR Studio Flow gather_input continuation

- This widget is used to gather_input's transition to next widget, upon detected speech

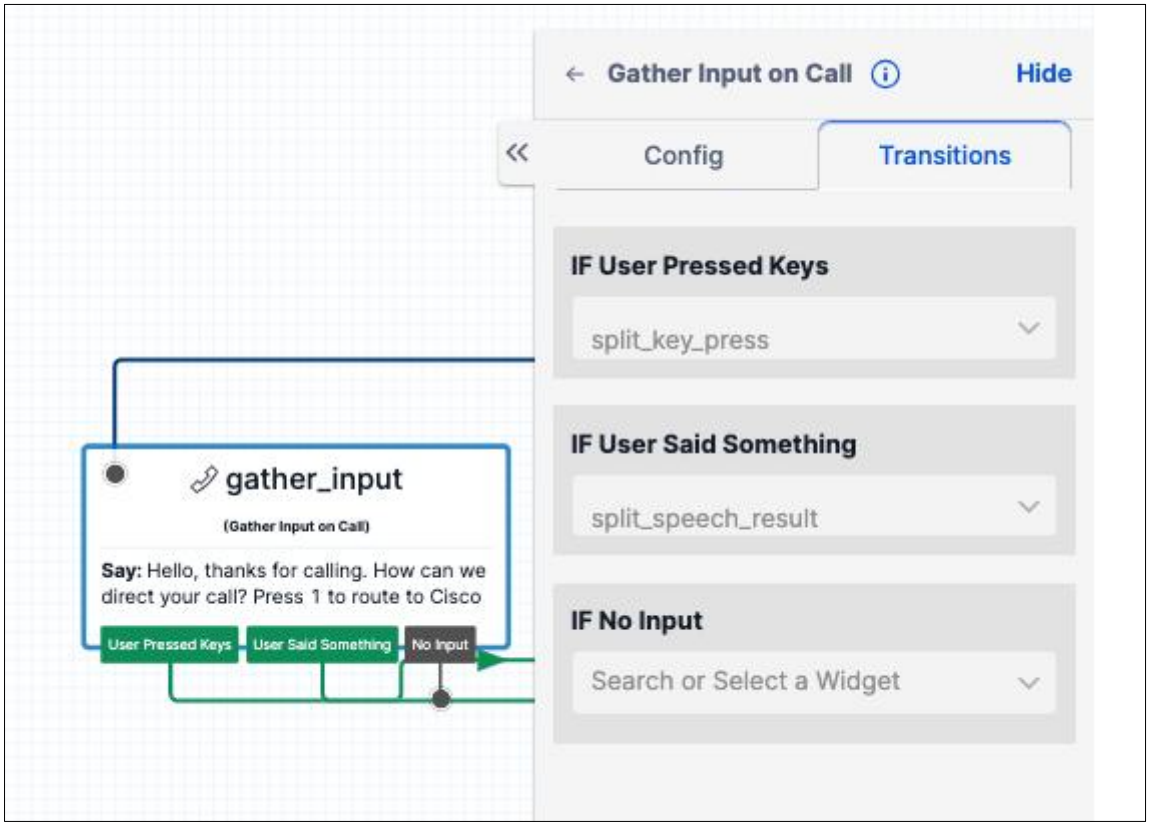

Figure 16 Twilio IVR Studio Flow gather_input continuation

Widget: split_speech

- This widget is configured to route any speech containing “Pizza” to the Pizza Ordering Virtual Agent Bot

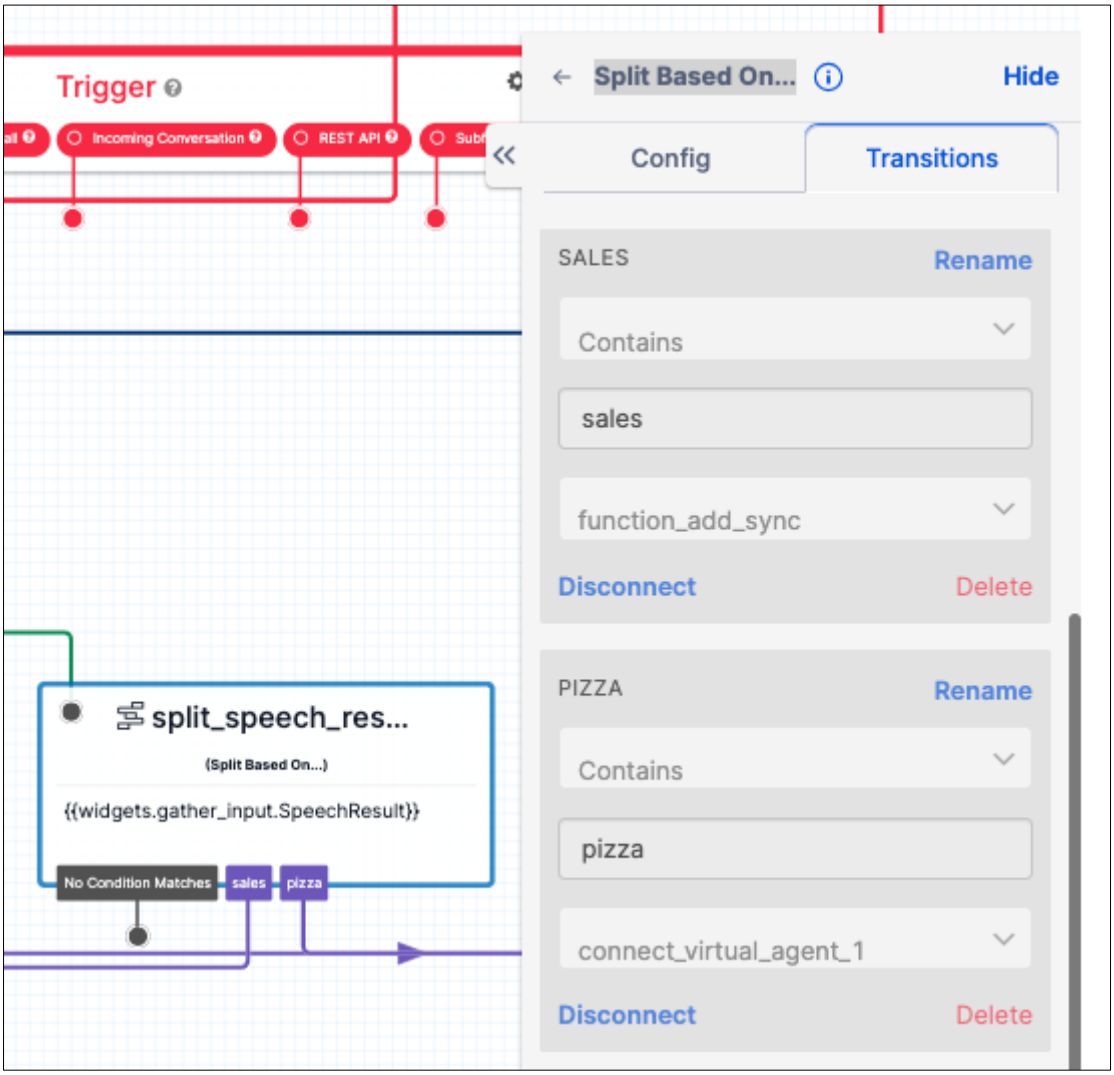

Figure 17 Twilio IVR Studio Flow – Split_Speech

Widget: Connect_to_VirtualAgent

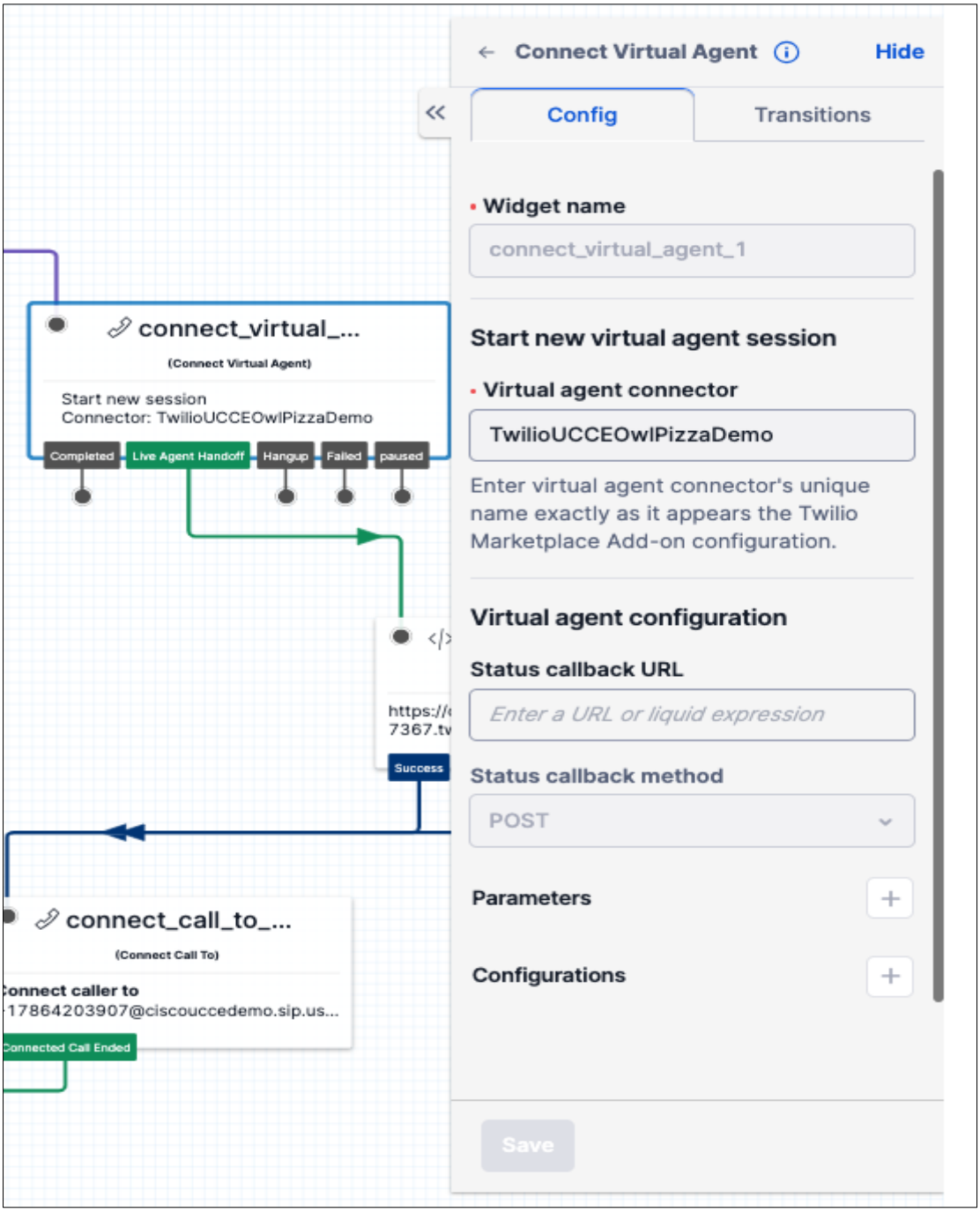

Figure 18 Twilio IVR Studio Flow – Connect_to_VirtualAgent

The Connect_to_Virtual Agent Widget is the widget put in place by the following Twilio and Google's One-Click Connector for Dialogflow and Twilio, as described in the Dialogflow Onboarding Guide (https://www.twilio.com/docs/voice/virtual-agent/dialogflow-cx-onboarding). Parameters and Bot configurations can be programmatically passed into to the Dialogflow bot via those entries here in this widget, as a way to pass in Unified Profile or other data input by the call in the <Gather> from Twilio/Studio, to the Google Dialogflow bot. Status callback webhooks can be sent to a customer's app, if that app needs to be aware of conversational turn-by-turn (intent-by-intent)

progress of the caller through the bot. Otherwise, action callbacks (with last matched intent or sentiment at last matched intent) can be returned when the caller exits the bot (change of state): e.g. at Live Agent Handoff, Session Paused, Hung up, etc.

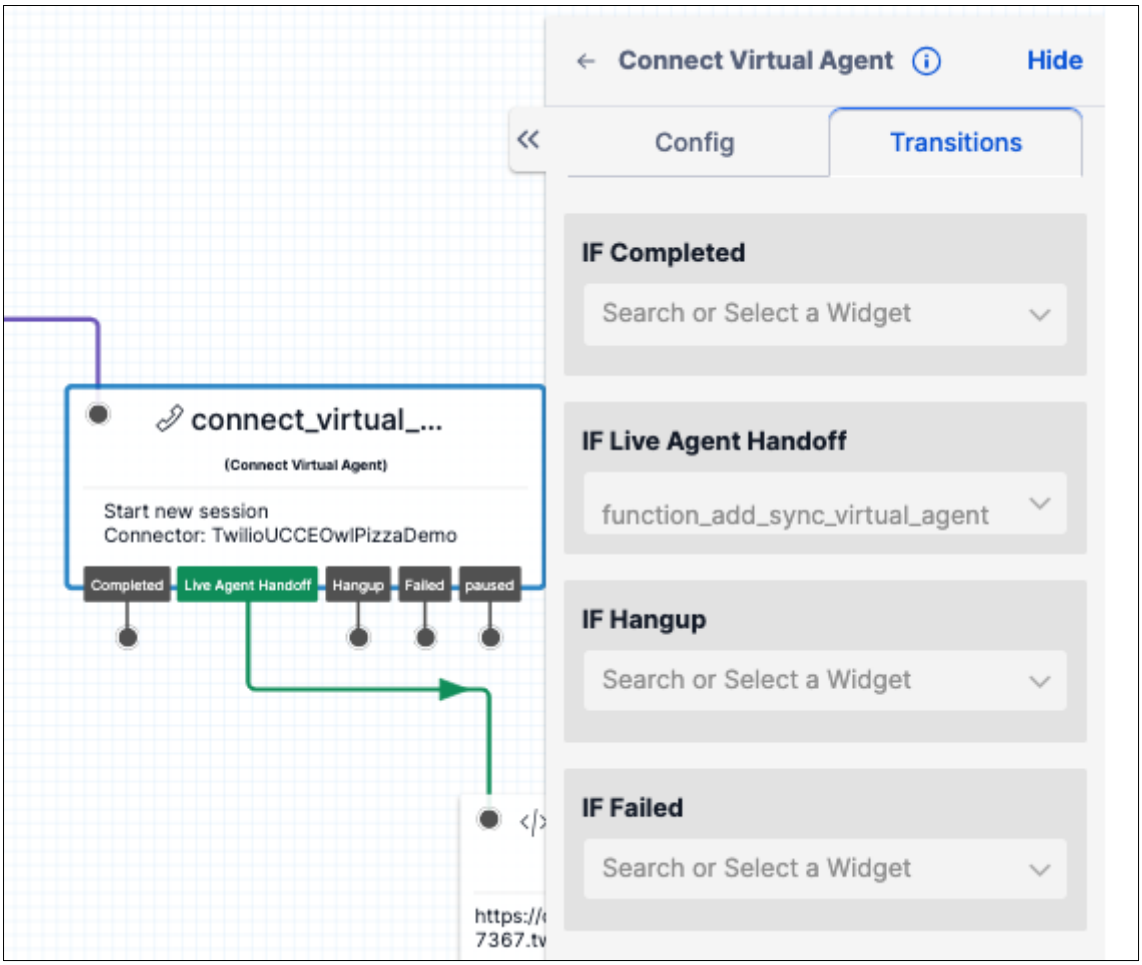

Figure 19 Twilio IVR Studio Flow – Connect_to_VirtualAgent continuation

The configured actions taken from the Dialogflow bot, depending upon which of 5 states the call exits the Dialogflow bot in are shown in the figure above. For this blueprint we've used Live Agent Handoff, but "paused" (the new Session Resumption feature of Dialogflow - https://www.twilio.com/docs/voice/twiml/connect/virtualagent/virtualagent-dialogflow-cx#pause-and-resume-a-conversation-session) could also be used (e.g. to resume a longer form/collection, after a live agent interaction, or to go off an make a payment and then return to the bot), or "Completed" to capture some input to the customers' profile from the bot (interaction). In this case, Live Agent Handoff then triggers a "function_add_sycn_virtual_agent" widget step.

Dialogflow CX Virtual Agent Configuration

Though much richer step-by-step directions for setting up Dialogflow and Twilio/Dialogflow integration are available at Google and here in the Twilio One-Click Integration Dialogflow On-boarding Guide (https://www.twilio.com/docs/voice/virtual-agent/dialogflow-cx-onboarding), the following summarizes the logical steps taken at the Dialogflow bot side, once that is set up and the call is passed to Dialogflow bot:

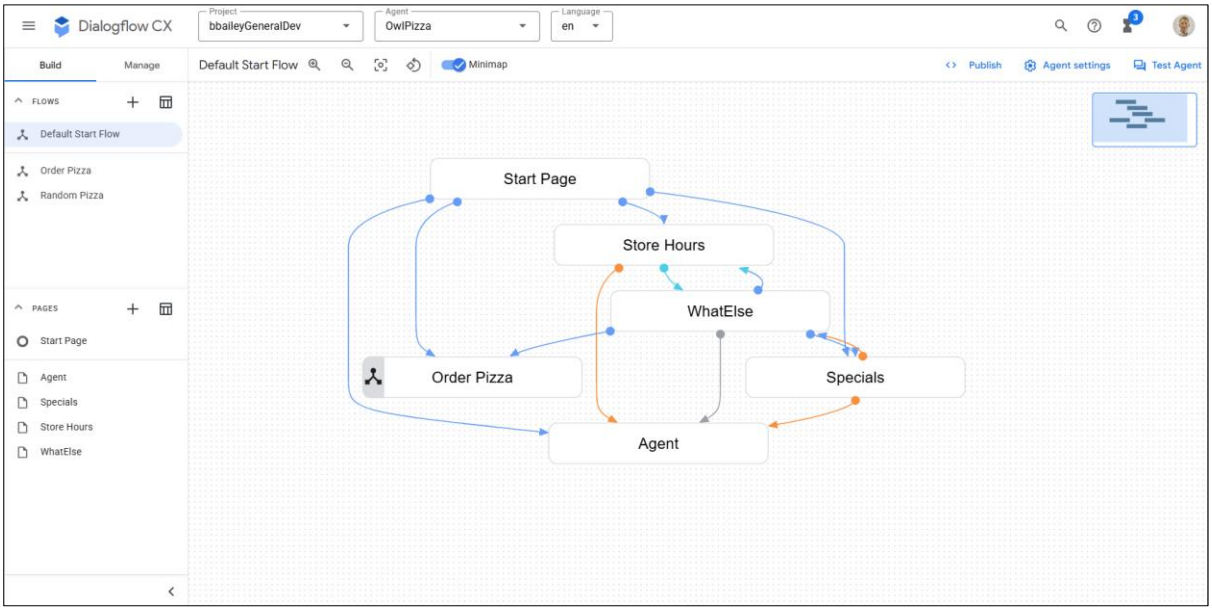

Figure 20 Basic Dialogflow Bot Setup

In Dialogflow CX a Start Page with Welcome Intent (first message /prompt played the caller if the caller says anything upon connection to the bot) is the logical starting point of the flow, followed by a series of detected "Intents" matched by the Dialogflow bot based on what the caller says to the bot (e.g. That they want to "Order a Pizza" or that they want to know the "Store Hours" or the "Specials")

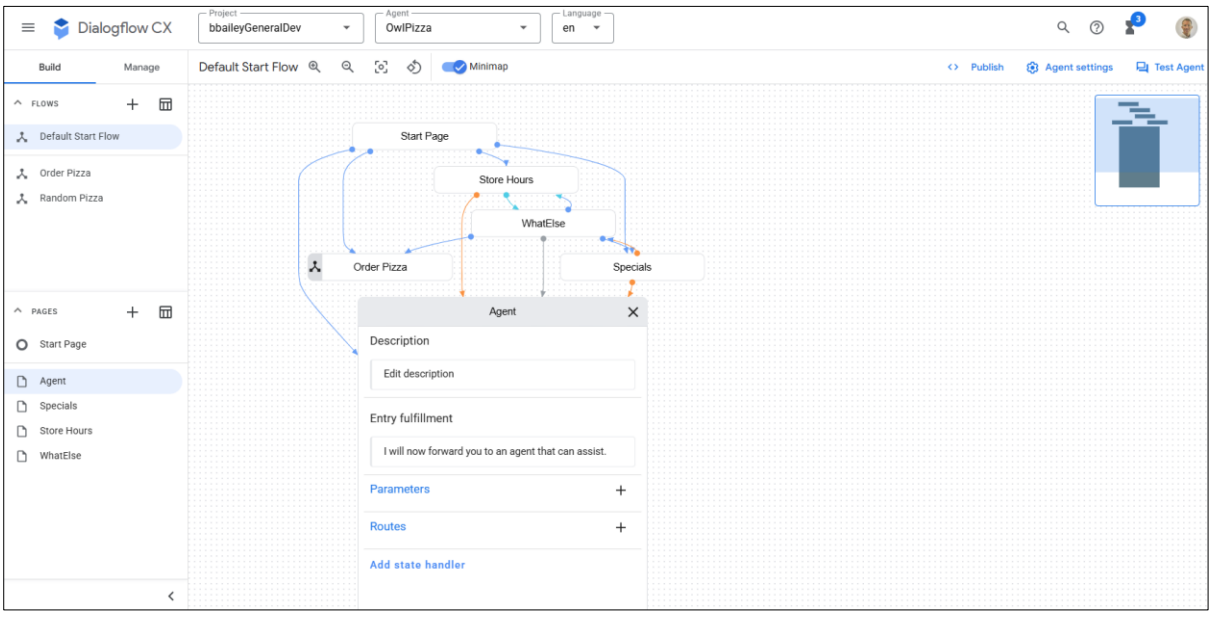

Figure 21 Dialogflow Bot setup continuation

When an intent is detected (spoken phrase matches an input/configured training phrase) a Fulfillment then describes the action taken (and text spoken to the caller as that action is taken – in this case forwarding the call to an Agent if, say, one of the specials are not available)

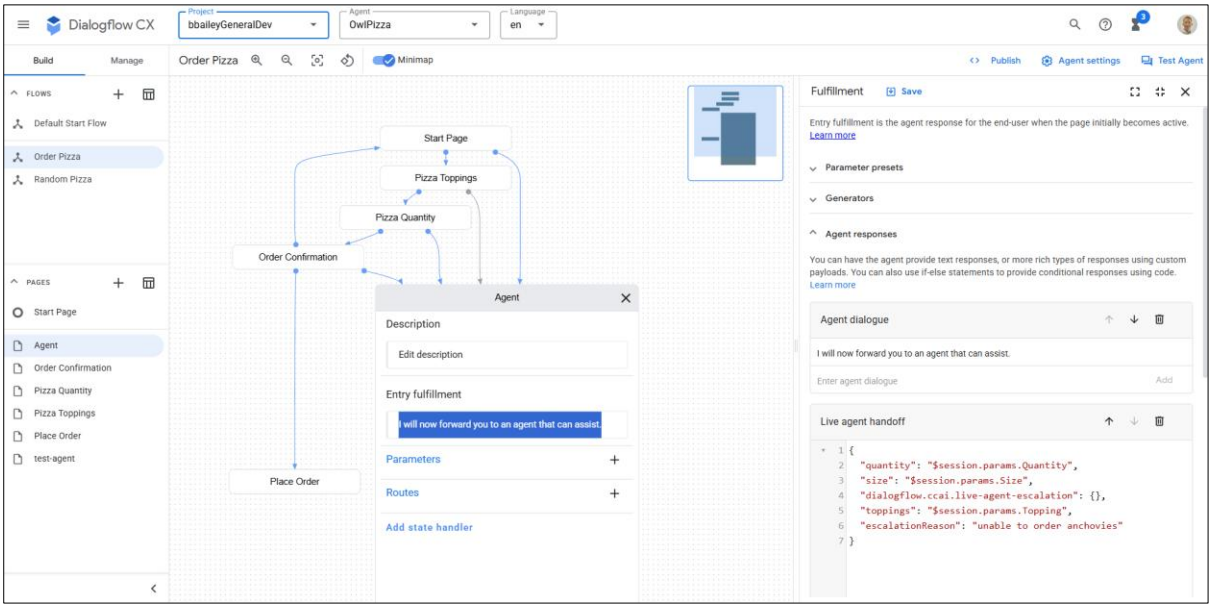

Figure 22 Dialogflow Bot setup continuation

When a call exits the bot – i.e. is forwarded by the bot to a human agent / Live Agent Escalation end state – (in this case due toppings unavailability), information about the caller's interaction with the bot is passed as a (red) set of parameters from the Google Dialogflow bot, back to the Twilio Platform (and from there in succeeding steps shown below, via Functions and Sync, to other downstream components that needs this Call Context information, like Cisco UCCE, and the Cisco Finesse Agent App / screen pop), so that the human agent getting handed the call can know the callers intent and what they said to the bot – and what triggered the escalation ("anchovies not available")… all without the caller having to repeat this information to the Agent.

Widget: function_add_sync

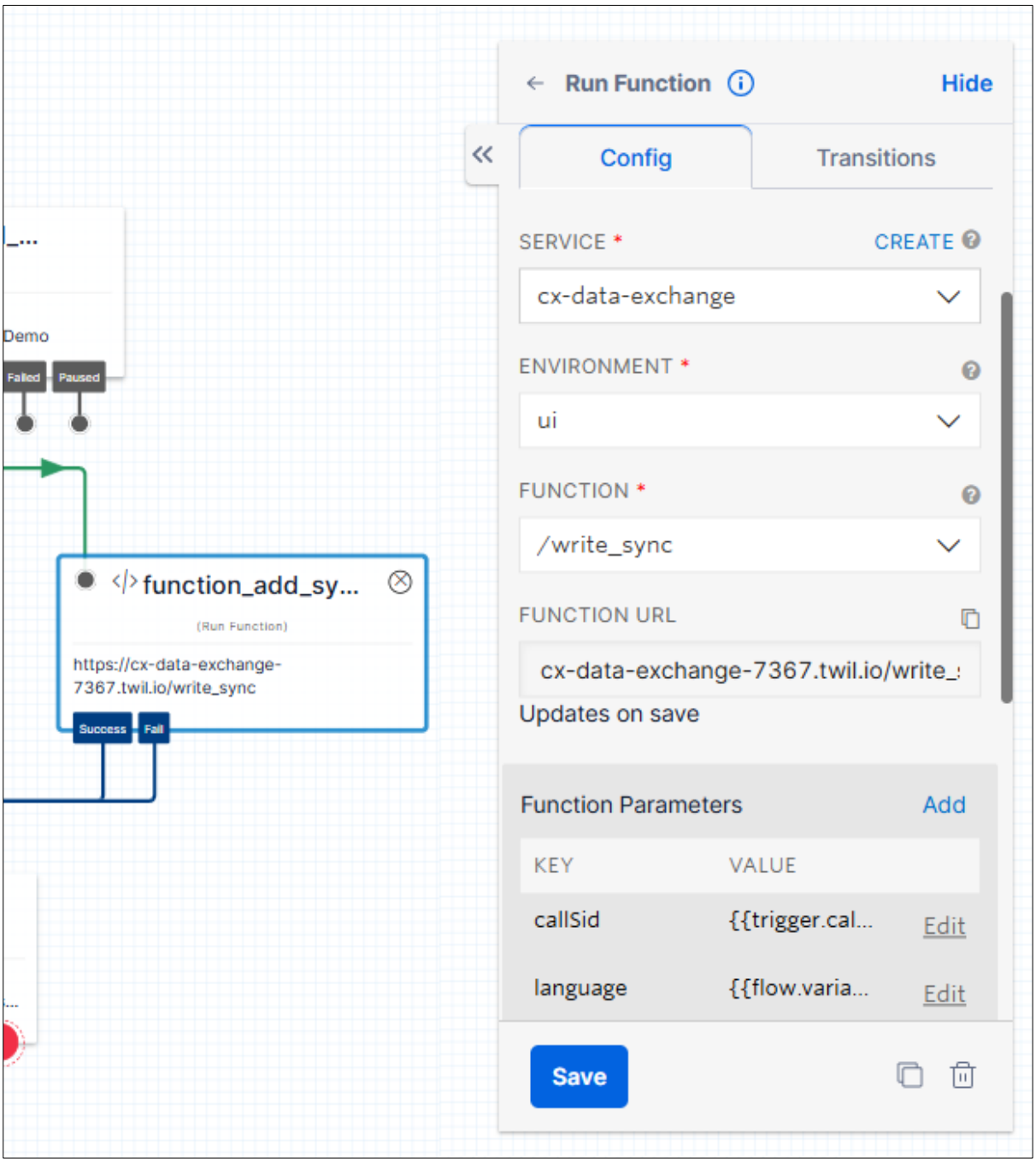

Figure 23 Twilio IVR Studio Flow – function_add_sync_virtual_agent

The function_add_sync widget takes the returned data from the Dialogflow CX Virtual agent (pizza toppings, quantity, etc.) and some of the original Twilio Studio data (caller language and phone number) and sets these variables in a Twilio Sync map via a custom Twilio Function.

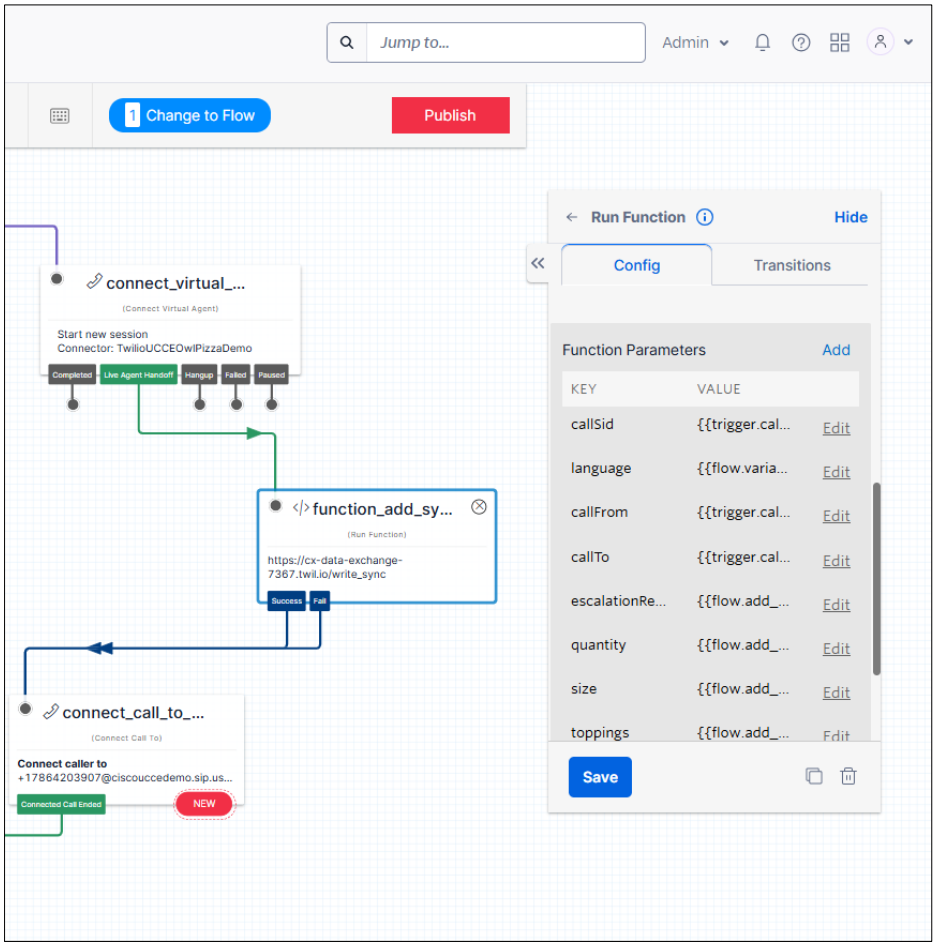

Figure 24 Twilio IVR Studio Flow – function_add_sync Continuation

The function_add_sync widget passes parameters from the call into Twilio Sync. The virtual agent connector will return the parameters set in the live agent handoff example, {{flow.add_ons.VirtualAgentData.VirtualAgentProviderData.AgentHandoffParameters.escalationReason}} the Twilio Studio flow simply sets them via a function (using sync). The original call sid is the key for the data in the sync map.

For more information on setting up the Function itself, see Section 5.1.6 below on setup of Twilio Functions.

Parameters that can be sent (via action callbacks for the other transition states besides Live Agent Handoff, as shown here) also include Last Matched Intent and Sentiment on Last Matched Intent, as well as many others. More info about these attributes passed and this newest feature of Twilio's programmatic integration with Dialogflow can be found here in the updated Twilio Virtual Agent documentation.

Widget: connect_call_to_sales

- In this widget the call is routed to Cisco UCCE for a live agent interaction.

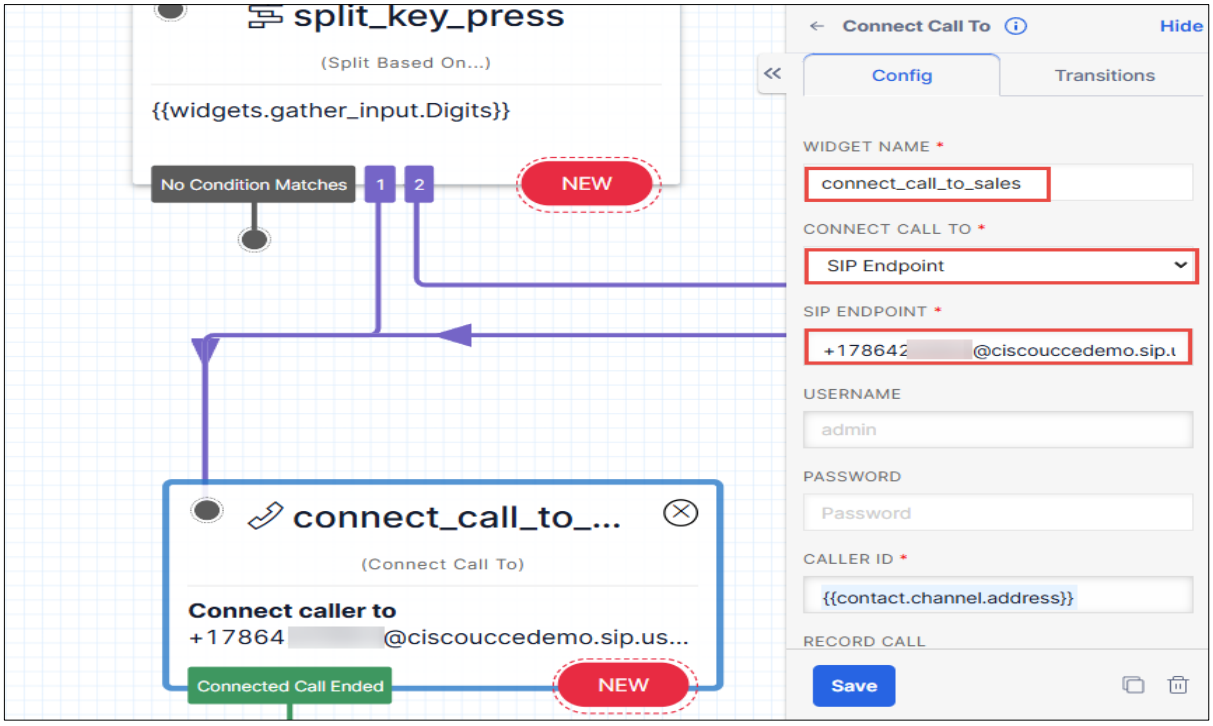

Figure 25 Twilio IVR Studio Flow – function_add_sync Continuation

If the call is not routed to UCCE, an error message is played as per configured in the wizard "say_play_1" as shown below.

Widget: say_play_1

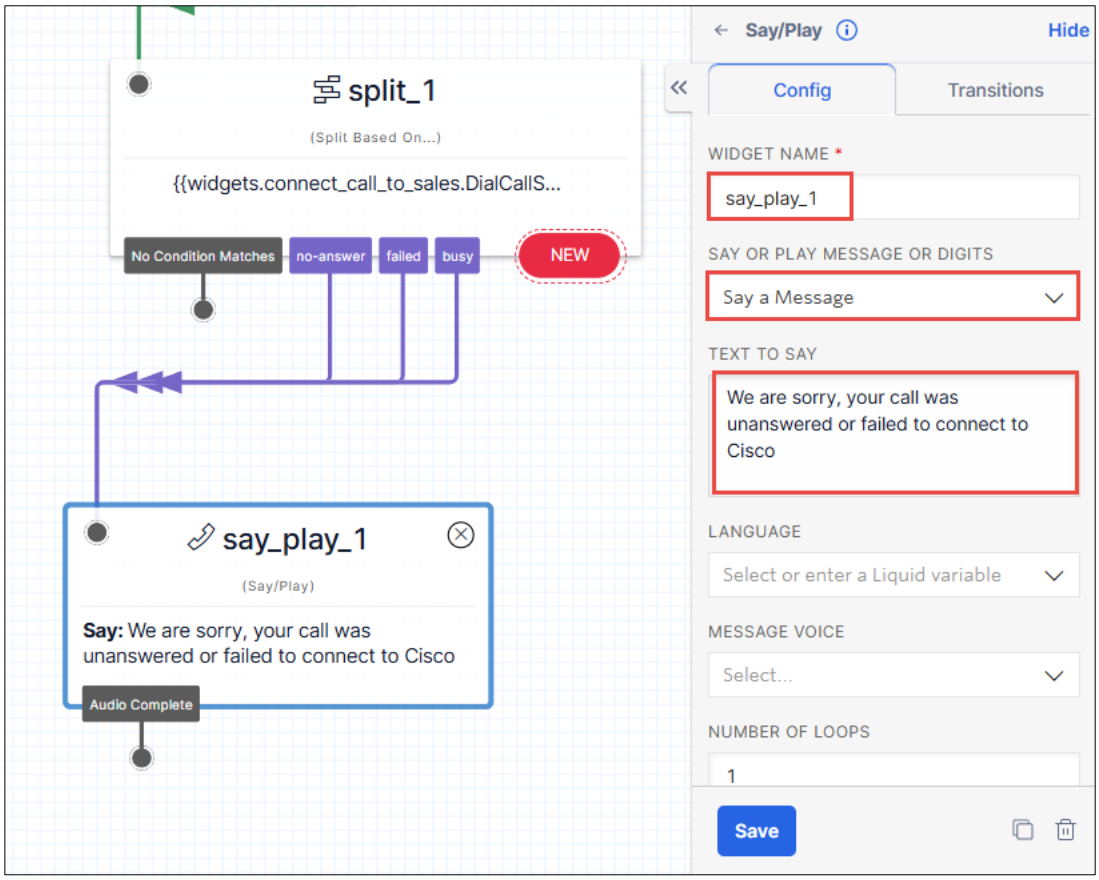

Figure 26 Twilio IVR Studio Flow – say_play_1

Widget: split_key_press

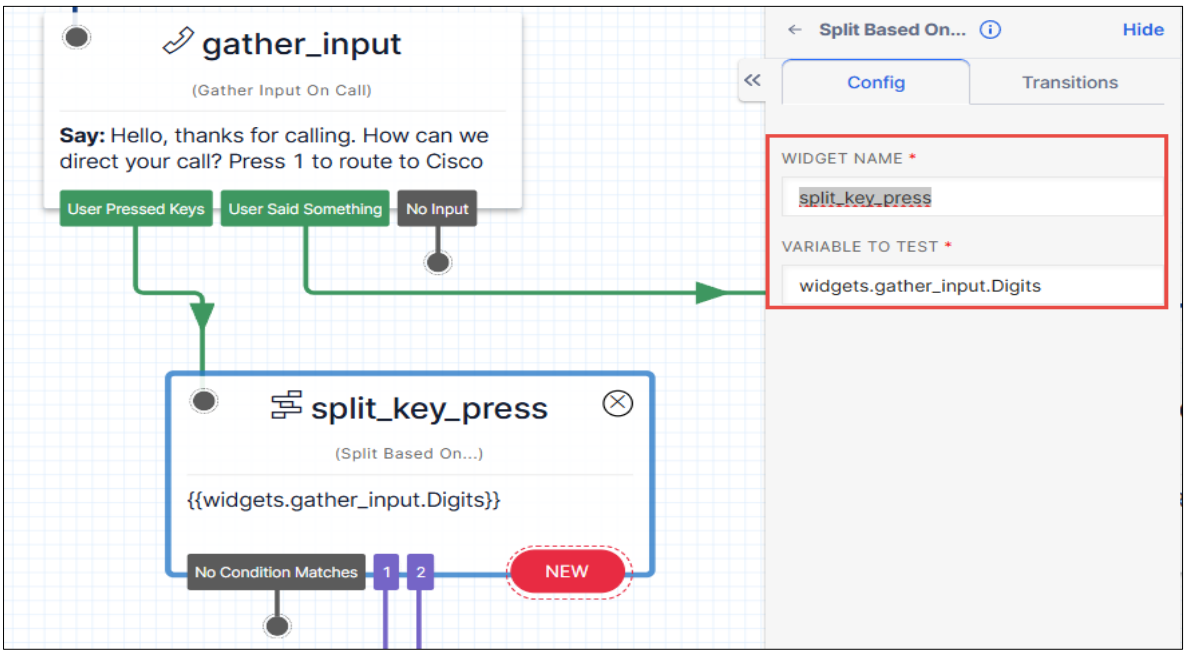

Figure 27 Twilio IVR Studio Flow – split_key_press

If the Caller provides DTMF 1 to continue to the UCCE flow, control will go to split_key_press nodes. (deployed here for flow validation and troubleshooting), i.e. to bypass Dialogflow for troubleshooting/development purposes.

5.1.5 Phone Numbers

- Navigate to **Phone Numbers > Manage > Active numbers**
- Under **Configure**,
    - Configure with: **Webhook, TwiMLBin, Function, Studio Flow, Proxy Service**
    - A call comes in: **Studio Flow**
    - Flow: **CiscoUCCEStudioFlow** (an IVR flow created in the Section 5.1.4)
    - Primary handler fails: **Webhook**
    - HTTP: **HTTP POST**

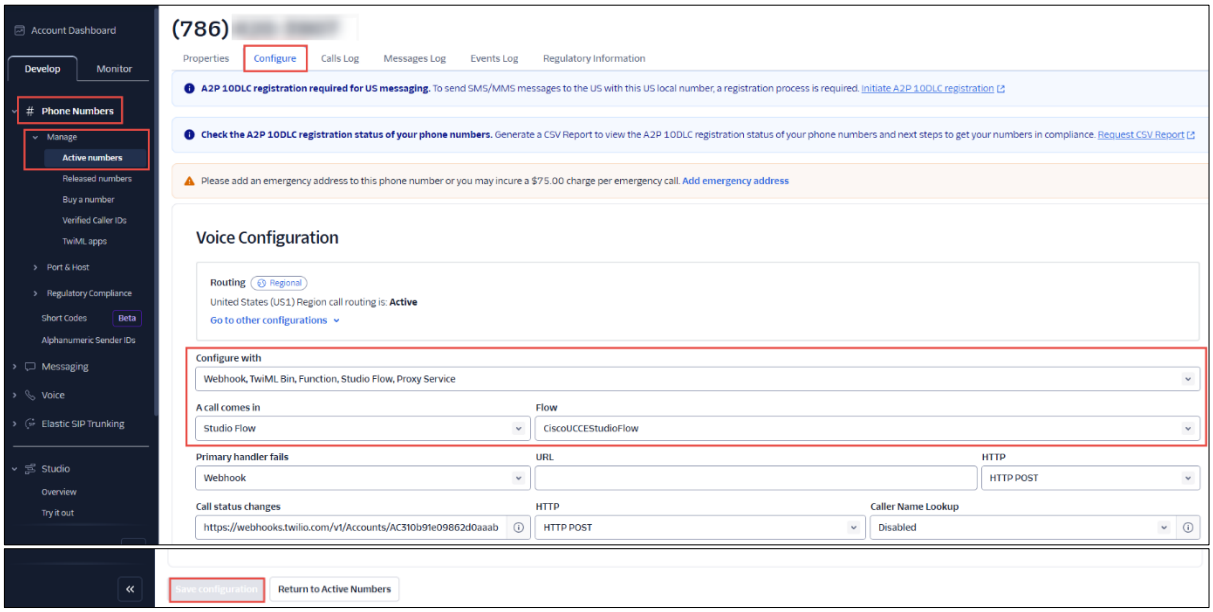

Figure 28 Phone Numbers

- Save configuration

5.1.6 Functions

- Navigate to **Explore Products > Functions and Assets > Services**
- Click **Create Service**

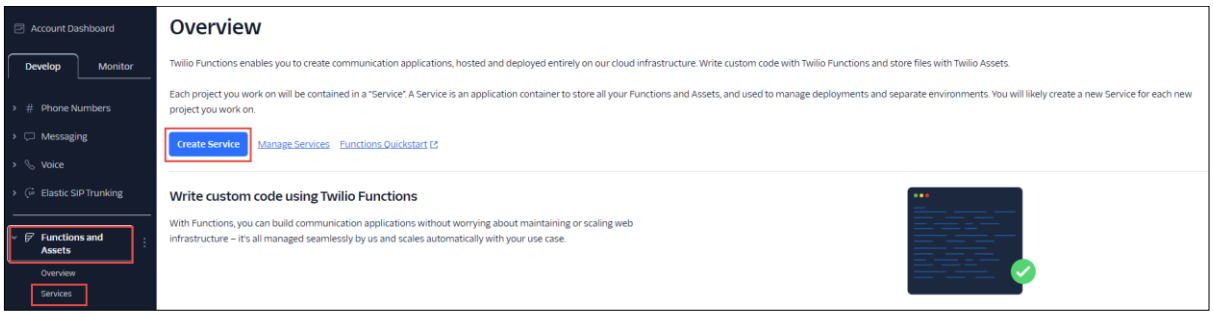

Figure 29 Functions – Create function

- Service Name: cx-data-exchange
- Click **Create your function**
- Functions: Type the function name e.g., **read_sync**
- Click Add+ to create more functions say, **write_sync**

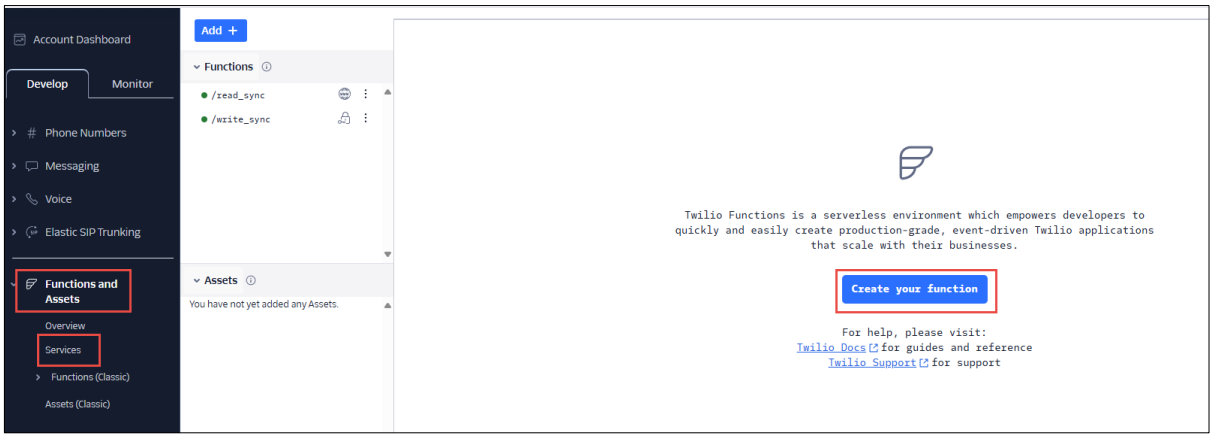

Figure 30 Create function continuation

- **read_sync** function is made as Public.
- Javascript for the read_sync function is shown in the right pane below

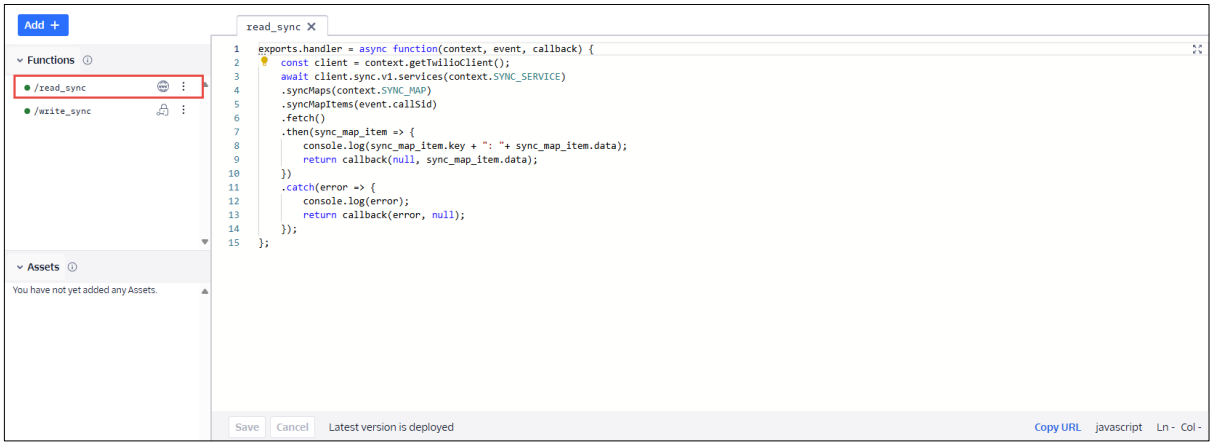

Figure 31 Functions – read_sync

- Click Add+ to create a function **write_sync**
- The URL for write sync function is https://cx-data-exchange-7367.twil.io/write_sync
- **write_sync** function is made Protected
- This function is used in the "function_add_sync" node of Twilio Studio IVR Flow

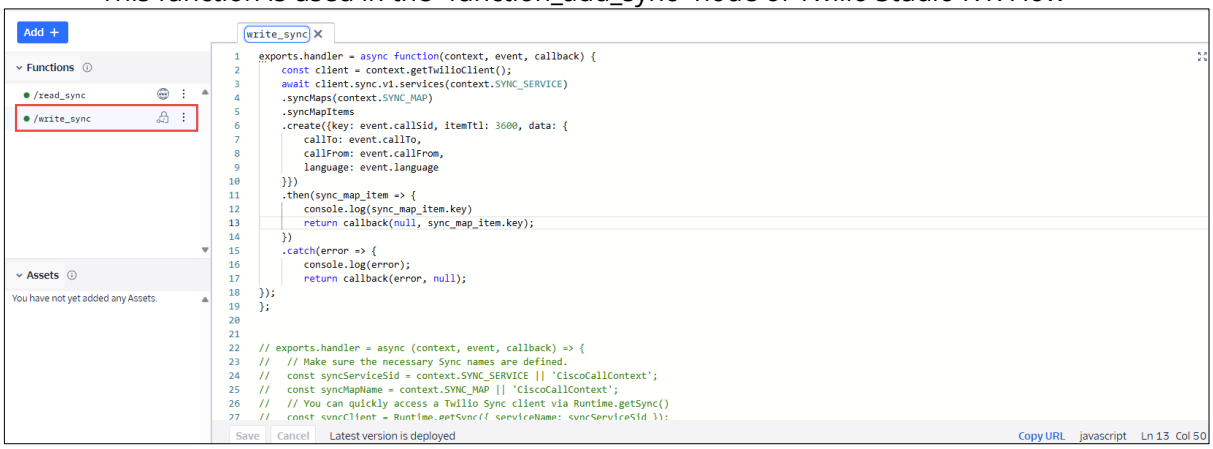

Figure 32 Functions – write_sync

The full text of the functions that are write_sync (private) and read_sync (public) is as follows:

write_sync:

```
exports.handler = async function(context, event, callback) {
   const client = context.getTwilioClient();
   await client.sync.v1.services(context.SYNC_SERVICE)
   .syncMaps(context.SYNC_MAP)
   .syncMapItems
   .create({key: event.callSid, itemTtl: 3600, data: {
      callTo: event.callTo,
      callFrom: event.callFrom,
      language: event.language,
      escalationReason: event.escalationReason,
      quantity: event.quantity,
      size: event.size,
      toppings: event.toppings
   }})
   .then(sync_map_item => {
      console.log(sync_map_item.key)
      return callback(null, sync_map_item.key);
   })
   .catch(error => {
      console.log(error);
      return callback(error, null);
-and-
```

read_sync:

```
exports.handler = async function(context, event, callback) {
   const client = context.getTwilioClient();
   await client.sync.v1.services(context.SYNC_SERVICE)
   .syncMaps(context.SYNC_MAP)
   .syncMapItems(event.callSid)
   .fetch()
   .then(sync_map_item => {
      console.log(sync_map_item.key + ": "+ sync_map_item.data);
      return callback(null, sync_map_item.data);
   })
   .catch(error => {
      console.log(error);
      return callback(error, null);
   });
};
```

- Then, navigate to **Settings & More > Environment Variables**
- Add the Key for **SYNC SERVICE and SYNC MAP**

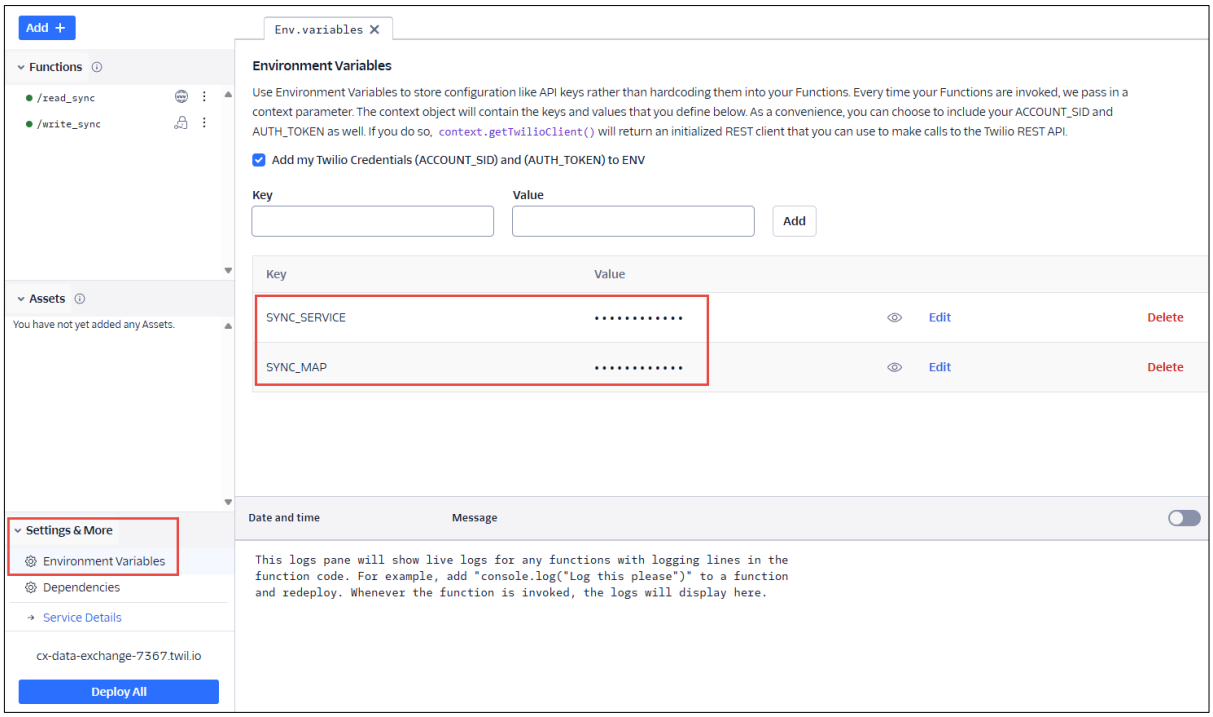

Figure 33 Functions – Environment Variables

Navigate to **Settings & More > Dependencies**

- Node Version**: Node.js v18**
- Below are the **Modules** and the corresponding **Versions** which are added as dependencies

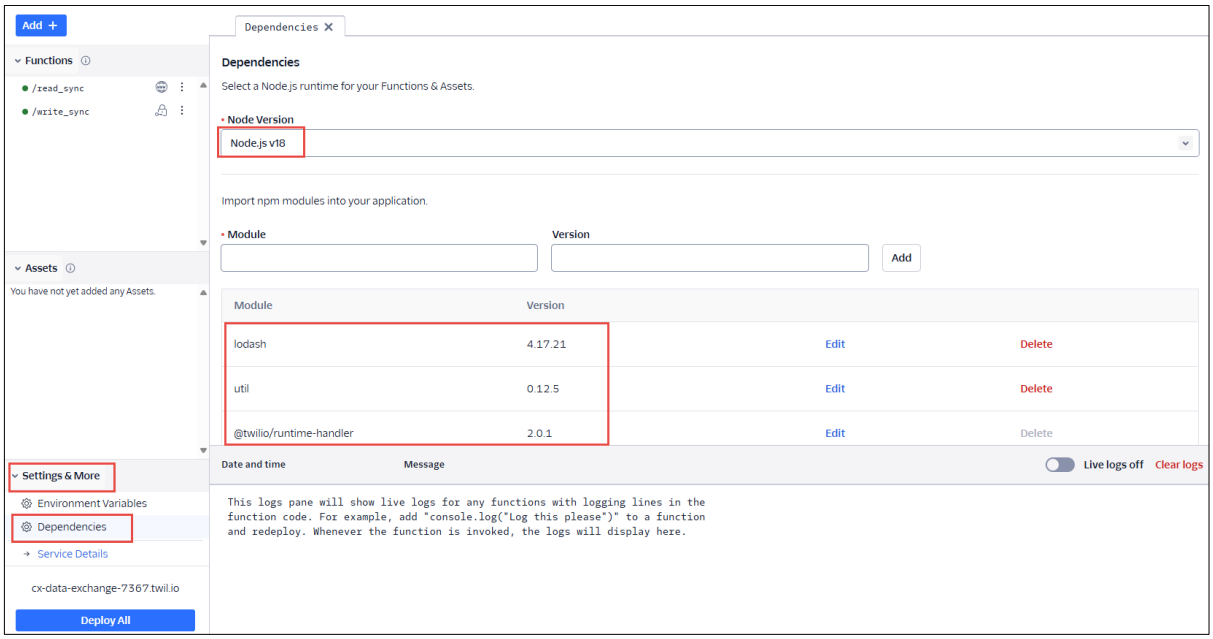

Figure 34 Functions – Dependencies

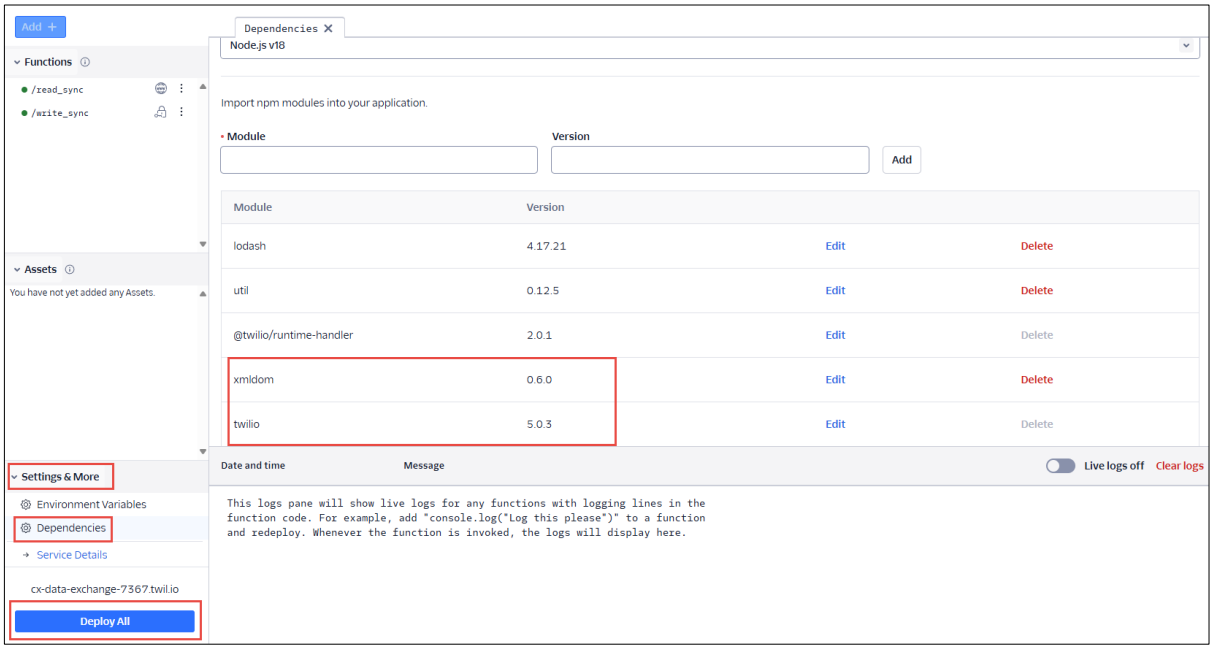

Figure 35 Functions – Dependencies continuation

- Click **Deploy All** to validate and deploy the code

5.1.7 Services

- Under Explore Products, navigate to **Sync > Services**
- Click **Create new Sync Service**
- Sync Service named **CiscoCallContext** is shown below
- Click **Save**

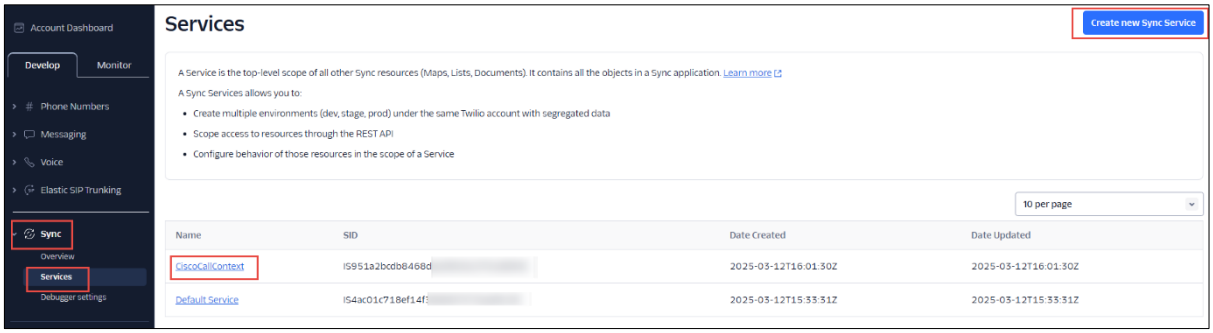

Figure 36 Sync Service

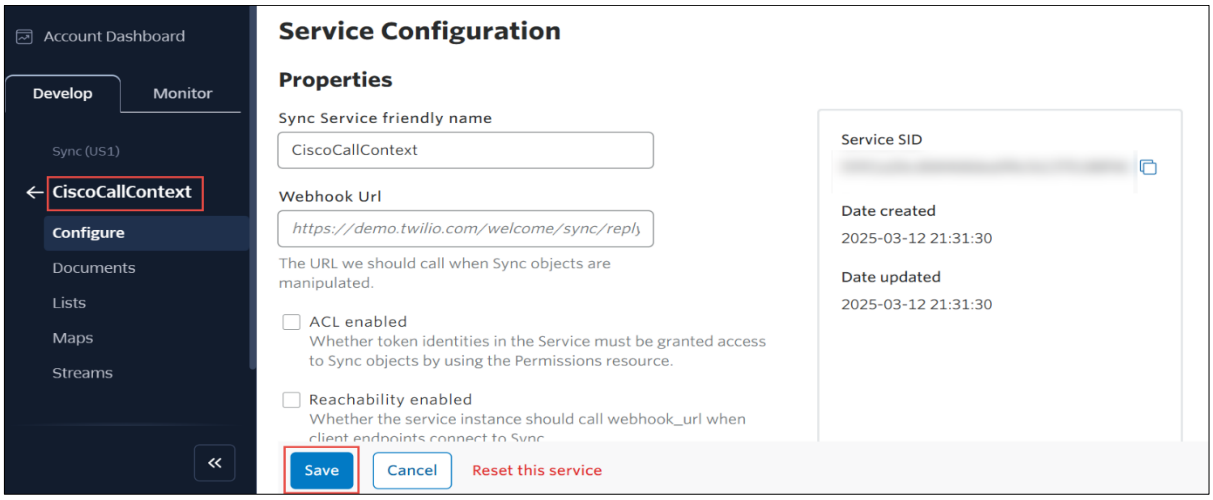

Figure 37 Sync Service Continuation

- Navigate to **CiscoCallContext > Maps**
- Click **Create new Sync Map**

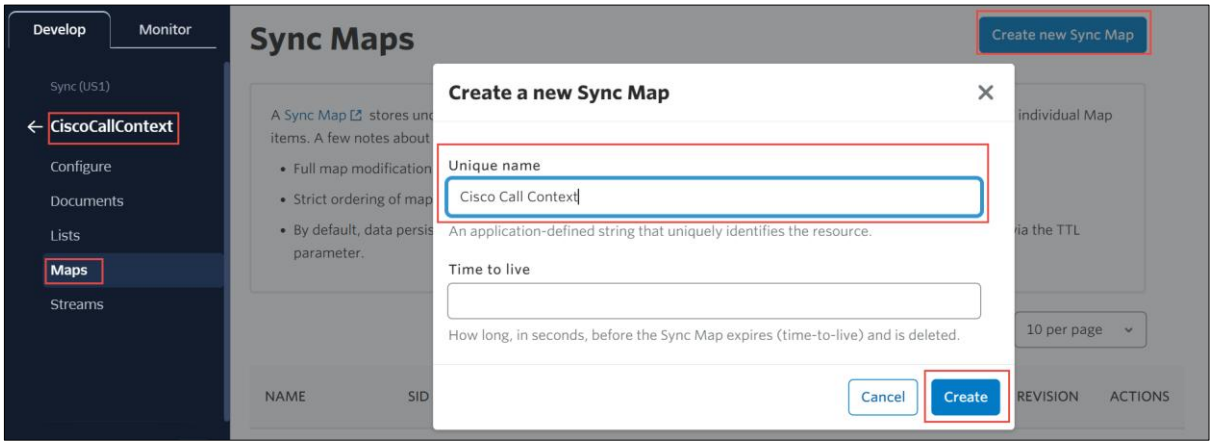

Figure 38 Create Sync Maps

- Sync Map named **CiscoCallContext** is shown below
- **KEY** value i.e., Twilio callSid and the **Map Item Data** are added to the database as shown below

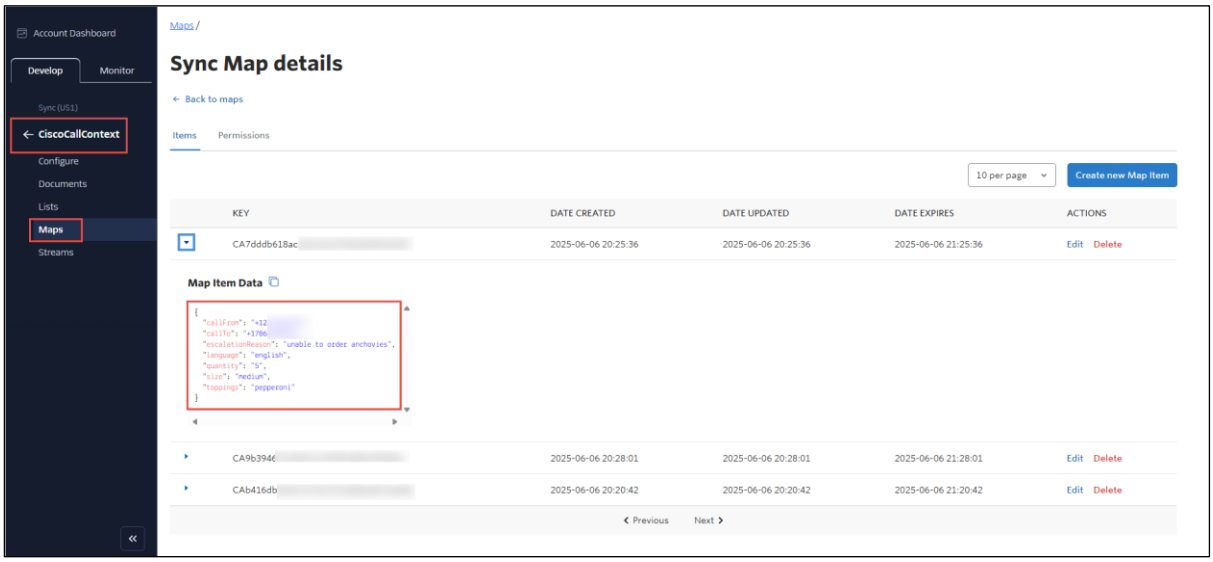

Figure 39 Sync Maps

5.2 Cisco UBE Configuration

Cisco Unified Border Element (CUBE) which is the Ingress gateway for UCCE environment includes the following configurations to route the incoming Twilio call to CVP.

5.2.1 Login to CUBE

Login to CUBE using Telnet or SSH. Use the IP address, username and password for the login.

5.2.2 Network Interface

Configure the network interface of the CUBE, one interface for LAN and one for WAN side. LAN side is configured towards CVP and WAN side towards Twilio.

```
interface GigabitEthernet0/0/0
 description Interface facing UCCE
 ip address 10.64.xx.xx 255.255.0.0
 negotiation auto

interface GigabitEthernet0/0/2
 description For Twilio
 ip address 192.65.xx.xx 255.255.255.128
 negotiation auto
```

5.2.3 Global CUBE Settings

Set the global configuration for CUBE as shown below:

```
voice service voip
 ip address trusted list
  ipv4 54.172.60.0 255.255.254.0
 address-hiding
 mode border-element
 allow-connections sip to sip
  trace
 sip
  session refresh
  asserted-id pai
```

5.2.4 Codecs

Configure the required codec profiles as shown below:

```
voice class codec 1
 codec preference 1 g711ulaw
 codec preference 2 g711alaw

voice class codec 2
 codec preference 1 g711ulaw
```

5.2.5 Voice Class Configurations

Voice class configurations for DPG (Dial Plan Group) and Tenant which is required by the Dial-Peers for call routing are configured here.

Voice Class DPG towards CVP

```
voice class dpg 500
 description Twilio to CVP
 dial-peer 3010 preference 1
```

Voice Class Tenant towards CVP

```
voice class tenant 100
 sip-server ipv4:172.16.XX.XX
 options-ping 60
 session transport tcp
 bind control source-interface GigabitEthernet0/0/0
 bind media source-interface GigabitEthernet0/0/0
```

Voice Class Tenant towards Twilio

```
voice class tenant 500
 registrar dns:XXXXXXXXdemo.sip.us1.twilio.com expires 600
 credentials username +17864XXXXXX password 6   AGXXXiLRIPEP\\XXXX realm sip.twilio.com
 authentication username +17864XXXXXX password 6
USXNV\OVZdRAD_OJKiMia]XXXXXXXXXXXX realm sip.twilio.com
 sip-server dns:XXXXXXXdemo.sip.us1.twilio.com
 options-ping 60
 session transport udp
 bind control source-interface GigabitEthernet0/0/2
 bind media source-interface GigabitEthernet0/0/2
```

Voice Class SIP OPTIONS

```
voice class sip-options-keepalive 900
 description For Ping
 transport udp
```

5.2.6 Dial Peers

Inbound and Outbound dial-peers are required in CUBE for the call routing between Twilio and CVP.

Inbound Dial-Peer for Twilio

```
dial-peer voice 1010 voip
 description *** Inbound Call from Twilio to CUBE-WAN ***
 session protocol sipv2
 destination dpg 500
 incoming called-number +17864XXXXXXX
 voice-class codec 1
 voice-class sip tenant 500
 dtmf-relay rtp-nte
 no vad
```

Outbound Dial-Peer for Twilio

```
dial-peer voice 2011 voip
 description *** Outbound Call from CUBE-WAN to Twilio****
 destination-pattern BAD.BAD
 session protocol sipv2
 session target sip-server
 voice-class codec 1
 voice-class sip tenant 500
 voice-class sip options-keepalive profile 900
 dtmf-relay rtp-nte
 no vad
```

Outbound Dial-Peer for CVP

```
dial-peer voice 3010 voip
 description ***Outbound to CVP from CUBE LAN for Twilio***
 translation-profile outgoing ucce23
 destination-pattern BAD.BAD
 session protocol sipv2
 session target sip-server
 voice-class codec 2
```

```
voice-class sip tenant 100
dtmf-relay rtp-nte
no vad
```

5.2.7 Voice Translation Rules

A voice translation rule is used to convert the incoming called number to UCCE configured Dialed Number which is required for the call routing.

Voice Translation Profile

```
voice translation-profile ucce23
 translate called 113
```

Voice Translation Rule

```
voice translation-rule 113
 rule 5 /^\+1786420XXXX/ /8500/
```

5.2.8 IP Route

Add the required IP route for Twilio network as shown below:

```
ip route 54.172.60.0 255.255.255.0 192.65.XX.XX
```

5.2.9 Check Trunk Status

```
cube7ucce#show sip-ua register status
Tenant:  500
---------- Registrar-Index  1 ----------
Line                    peer     expires(sec) reg survival
P-Asocial-U                                              RI
=============================== ========= ====
+1786420XXXX              -1       460       yes normal
```

5.2.10 Cisco UBE Running Configuration

```
cube7ucce#sh running-config br
Building configuration...
Current configuration : 24214 bytes
!
version 17.12
service config
service timestamps debug datetime msec
service timestamps log datetime msec
service call-home
platform qfp utilization monitor load 80
platform punt-keepalive disable-kernel-core
!
hostname cube7ucce
!
boot-start-marker
boot system flash:isr4300-universalk9.17.12.04a.SPA.bin
boot-end-marker
!
vrf definition Mgmt-intf
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
!
logging buffered 147483647
aaa new-model
!
vtp version 1
!
multilink bundle-name authenticated
!
password encryption aes
!
!
voice service voip
 ip address trusted list
  ipv4 54.172.60.0 255.255.254.0
address-hiding
 mode border-element
 allow-connections sip to sip
 fax protocol pass-through g711ulaw
 trace
 sip
  session refresh
```

```
  asserted-id pai
!
voice class codec 1
 codec preference 1 g711ulaw
 codec preference 2 g711alaw
!
voice class codec 2
 codec preference 1 g711ulaw
!
voice class dpg 500
 description Twilio to CVP
 dial-peer 3010 preference 1
!
voice class sip-options-keepalive 900
 description For Ping
 transport udp
!
voice class tenant 100
 sip-server ipv4:172.16.XX.XX
 options-ping 60
 session transport tcp
 bind control source-interface GigabitEthernet0/0/0
 bind media source-interface GigabitEthernet0/0/0
!
voice class tenant 500
 registrar dns:XXXXXXXXdemo.sip.us1.twilio.com expires 600
 credentials username +17864XXXXXX password 6   AGXXXiLRIPEP\\XXXX realm sip.twilio.com
 authentication username +17864XXXXXX password 6
USXNV\OVZdRAD_OJKiMia]XXXXXXXXXXXX realm sip.twilio.com
 sip-server dns:XXXXXXXdemo.sip.us1.twilio.com
 options-ping 60
 session transport udp
 bind control source-interface GigabitEthernet0/0/2
 bind media source-interface GigabitEthernet0/0/2
!
voice translation-rule 113
 rule 5 /^\+17864203XXX/ /8500/
!
voice translation-profile ucce23
 translate called 113
!
diagnostic bootup level minimal
!
no license feature hseck9
license udi pid ISR4331/K9 sn XXXXXXXXXXXXX
license boot level appxk9
license boot level uck9
license boot level securityk9
memory free low-watermark processor 368651
```

```
!
spanning-tree extend system-id
!
redundancy
 mode none
!
interface GigabitEthernet0/0/0
 description CUBE 7 Interface facing UCCE
 ip address 10.64.XX.XX 255.255.0.0
 negotiation auto
!
interface GigabitEthernet0/0/2
 description CUBE 7 Public
 ip address 192.65.XX.XX 255.255.255.XX
 negotiation auto
!
negotiation auto
!
interface Service-Engine0/2/0
!
interface Service-Engine0/4/0
!
interface GigabitEthernet0
 vrf forwarding Mgmt-intf
 no ip address
 negotiation auto
!
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip http client source-interface GigabitEthernet0/0/0
!
ip rtcp report interval 2000
ip route 0.0.0.0 0.0.0.0 10.64.1.1
ip route 54.172.60.0 255.255.255.0 192.65.XX.XX
ip route 172.16.0.0 255.255.0.0 10.64.XX.XX
ip ssh bulk-mode 131072
!
control-plane
!
voice-port 0/2/0
!
voice-port 0/2/1
!
dial-peer voice 1010 voip
 description *** Inbound Call from Twilio to CUBE-WAN ***
 session protocol sipv2
 destination dpg 500
```

```
 incoming called-number +1786420XXXX
 voice-class codec 1
 voice-class sip tenant 500
 dtmf-relay rtp-nte
 no vad
!
dial-peer voice 2011 voip
 description *** Outbound Call from CUBE-WAN to Twilio****
 destination-pattern BAD.BAD
 session protocol sipv2
 session target sip-server
 voice-class codec 1
 voice-class sip tenant 500
 voice-class sip options-keepalive profile 900
 dtmf-relay rtp-nte
 no vad
!
dial-peer voice 3010 voip
 description ***Outbound to CVP from CUBE LAN for Twilio***
 translation-profile outgoing ucce23
 destination-pattern BAD.BAD
 session protocol sipv2
 session target sip-server
 session transport tcp
 voice-class codec 2
 voice-class sip tenant 100
dtmf-relay rtp-nte
 no vad
!
line con 0
 exec-timeout 5 0
 password 6 XXXXXXX
 logging synchronous
 stopbits 1
line aux 0
line vty 0 4
 exec-timeout 15 0
 password 6 XXXXXXX
 logging synchronous
transport input telnet
line vty 5 14
 transport input ssh
!
  active
  destination transport-method http
!
end
```

5.3 CVP Configuration

This section provides the configuration required in CVP to pass the SIP Header "x-ciscodata" sent from Twilio to Cisco UCCE environment. Prior to this, CVP should have the basic configurations required for the integration and call routing with the other components of UCCE.

5.3.1 Configure SIP Header Passing in CVP

- Login to CVP OAMP Server
- Login to Cisco Unified Customer Voice Portal by providing the Username and Password

Figure 40 CVP OAMP login

- Navigate to **Device management** > **Unified CVP Call Server**

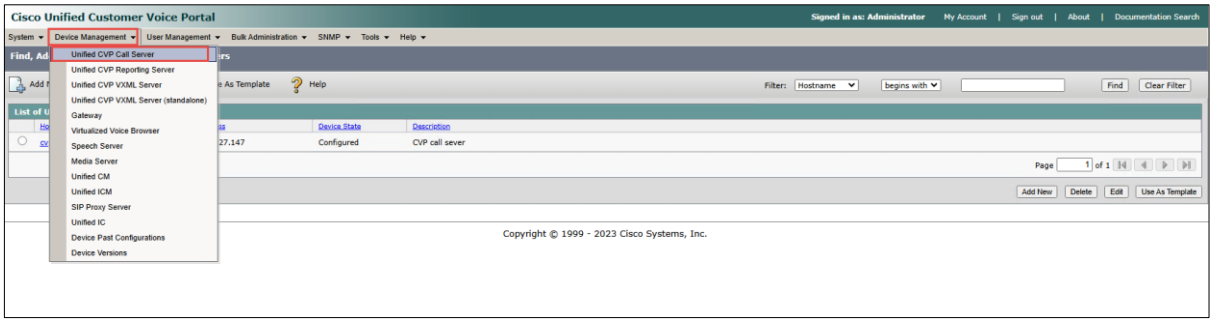

Figure 41 CVP Call Server

- Select and open the CVP call server from the list

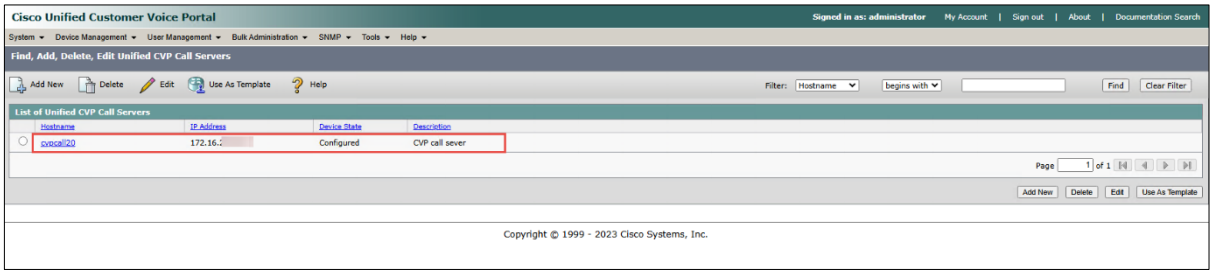

Figure 42 Open CVP Call Server

- Choose **SIP settings** and open **Advanced Configuration**

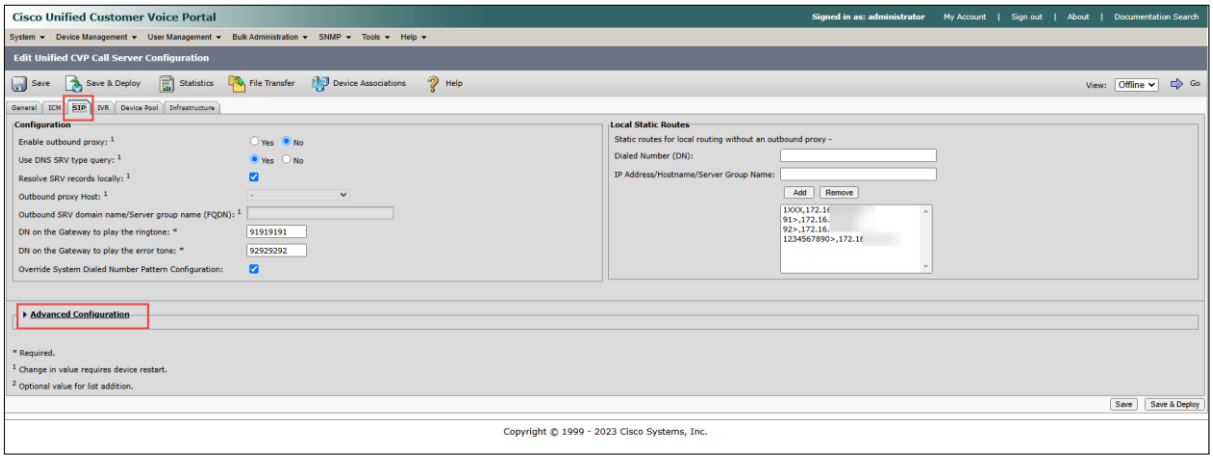

Figure 43 CVP SIP Configuration

- In **Advanced Configuration** move down to **SIP Header Passing (to ICM)** section.
- Add the header name "x-ciscodata" which is sent by Twilio for passing the Call ID.
- Provide the **Header Name** and Click **Add**

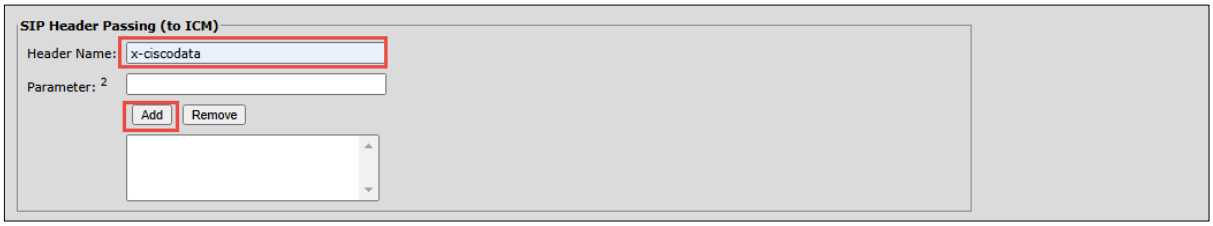

Figure 44 Adding SIP Header in CVP

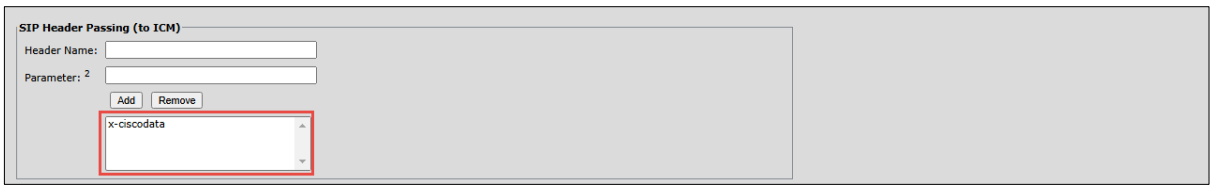

Figure 45 Added SIP Header in CVP

- After adding the SIP Header, **Save and Deploy** the configuration.

Figure 46 Save and Deploy

- Navigate to **System** > **Control Center**
- Check the service status of CVP Call Server. It should be Up

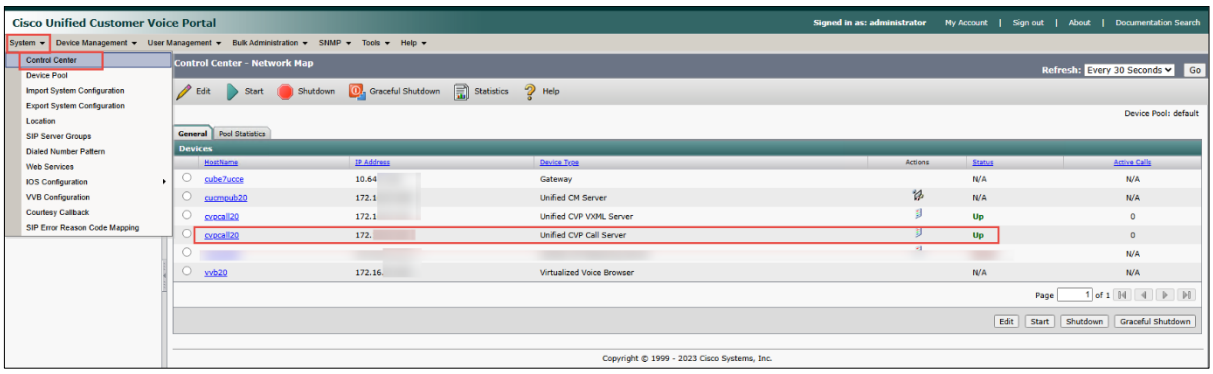

Figure 47 Check CVP Call Server Status

5.4 Cisco ICM Configuration

This section provides the ICM Script configuration based on which the Inbound call is routed to the available Skill Group Agent.

5.4.1 ICM Script Configuration

- Login to Cisco ICM Server
- Open **Cisco Unified CCE Tools** and navigate to **Administration tools** and open **Script Editor**
- A basic Inbound call flow is created in ICM, which prompts the caller to enter a selection for a skill group of their choice.

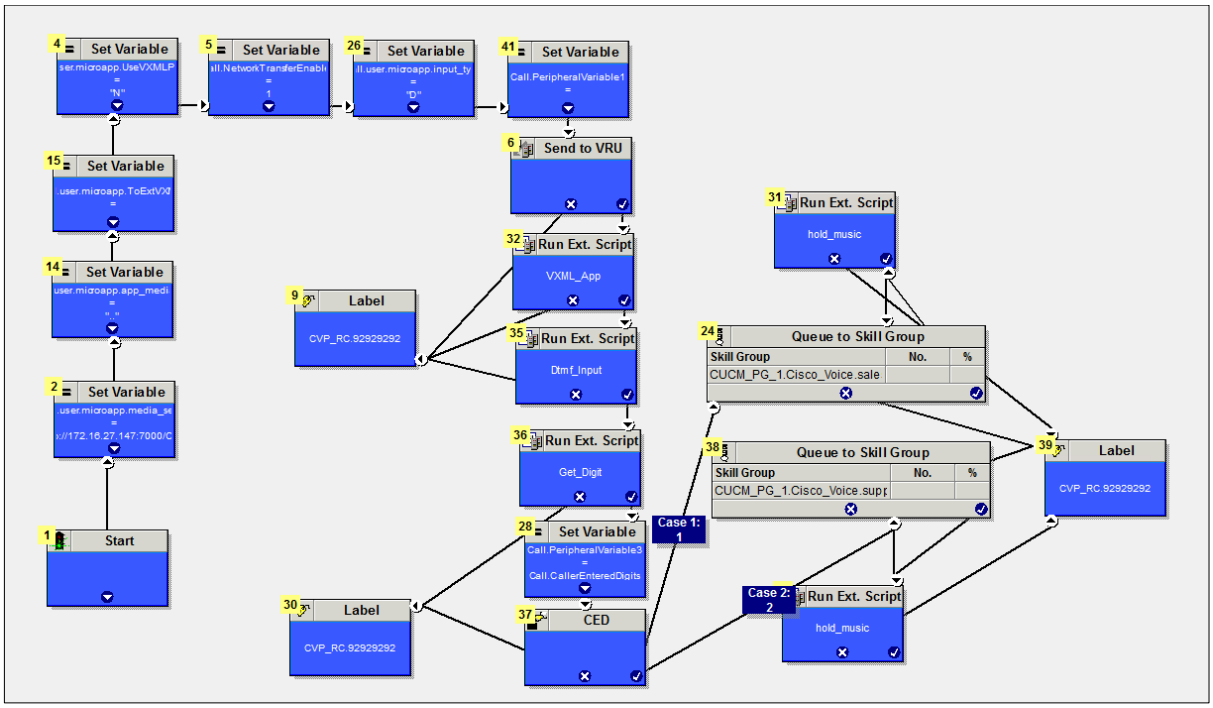

Figure 48 ICM Script (Script name: cvp_dtmf_Twilio)

- In the flow configuration, a peripheral variable is configured to get the Twilio Call ID send in the SIP Header (x-ciscodata) of the call Invite using the value "**substr(Call.SIPHeader,13,(len(Call.SIPHeader)))"**

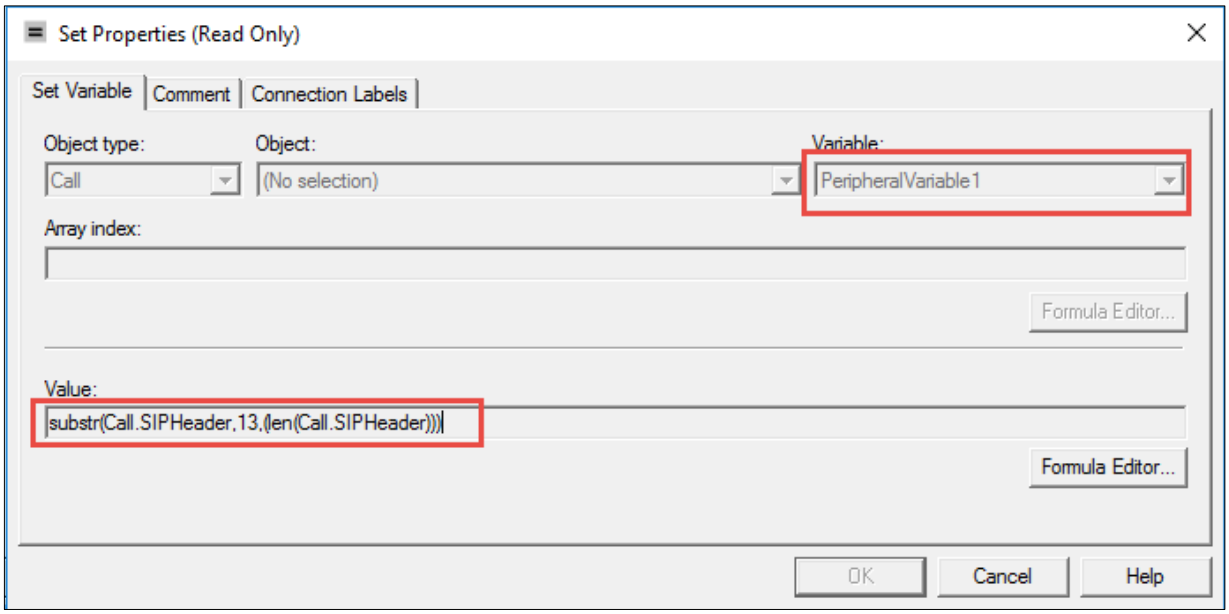

Figure 49 ICM Script – Peripheral Variable

5.4.2 Dialed Number and Call Type

- Open **Cisco Unified CCE Tools** in ICM Server and navigate to **Administration tools** and open **Configuration Manager**
- Expand **Tools** > **List Tools** > and select **Call Type List**

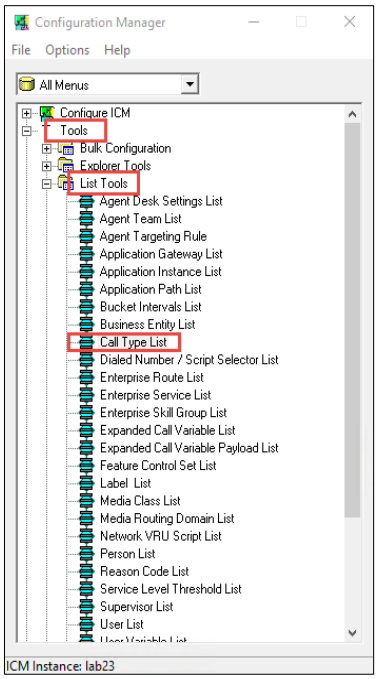

Figure 50 ICM Configuration Manager

- Click **Retrieve**. CVP2_CT is used for this configuration.

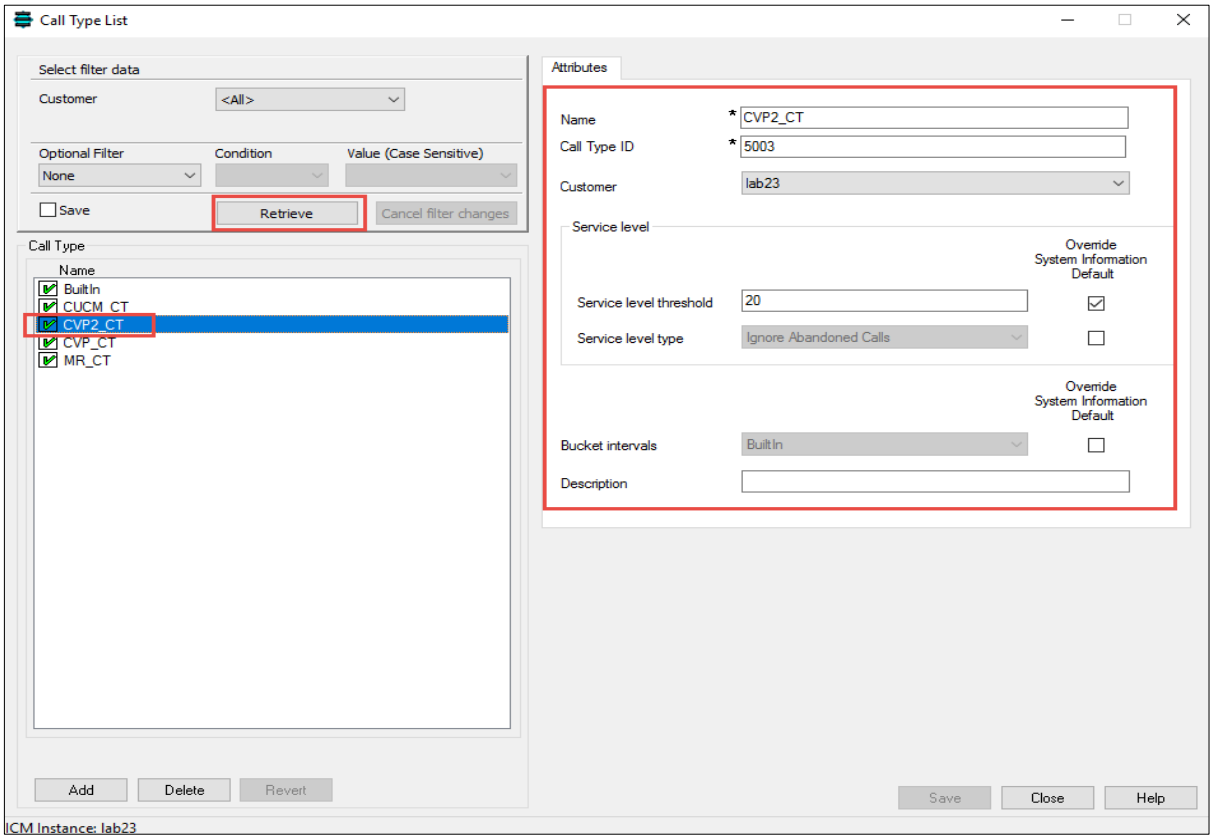

Figure 51 ICM Call Type

- In Configuration Manager, Expand **Tools** > **List Tools** > and select **Dialed Number / Script Selector List**

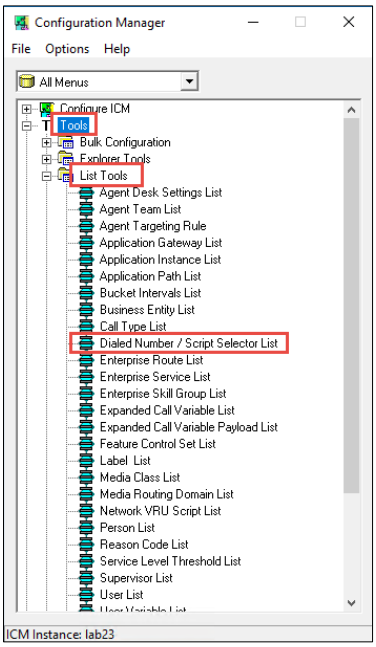

Figure 52 ICM Dialed Number / Script Editor

- Click Retrieve. Dialed Number 8500 is used for this configuration

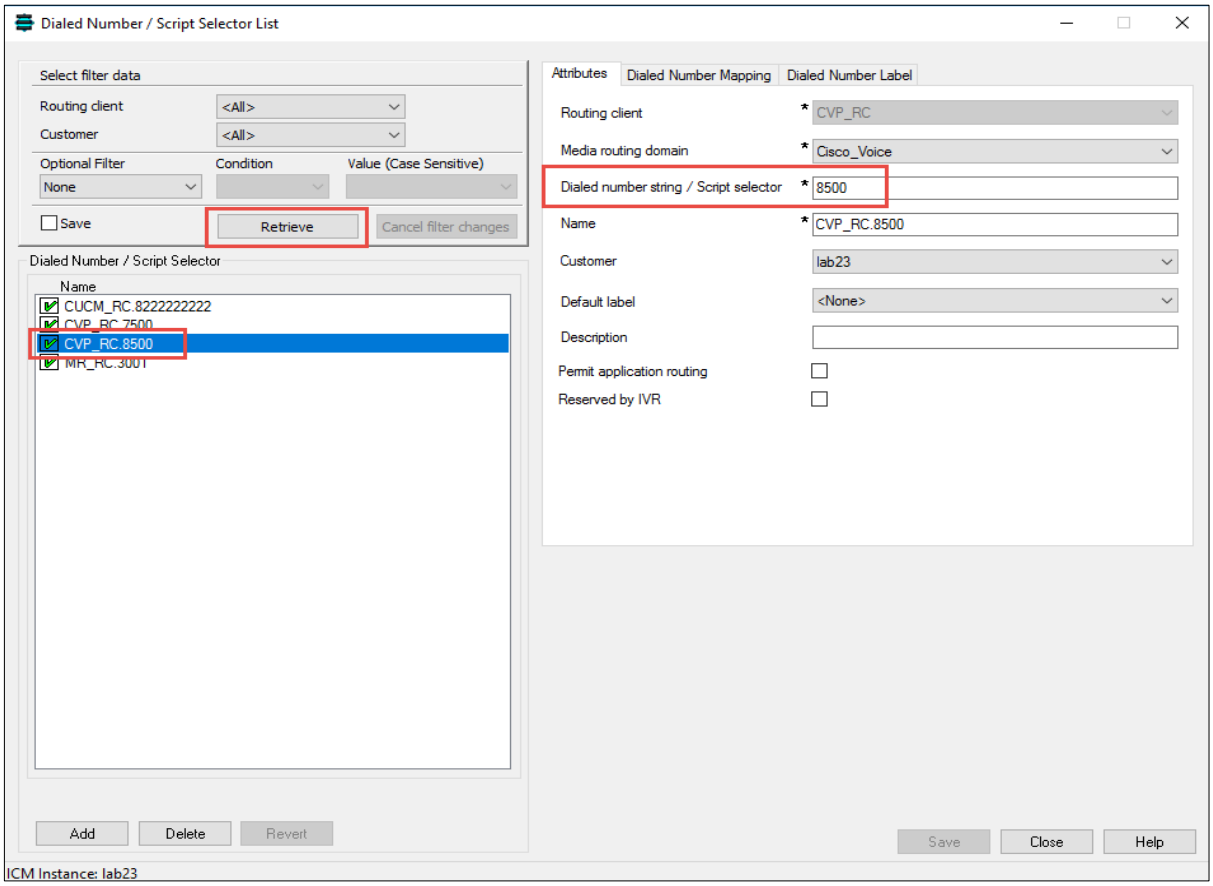

Figure 53 ICM Dialed Number / Script Editor Continuation

- Map the Dialed Number with Call Type

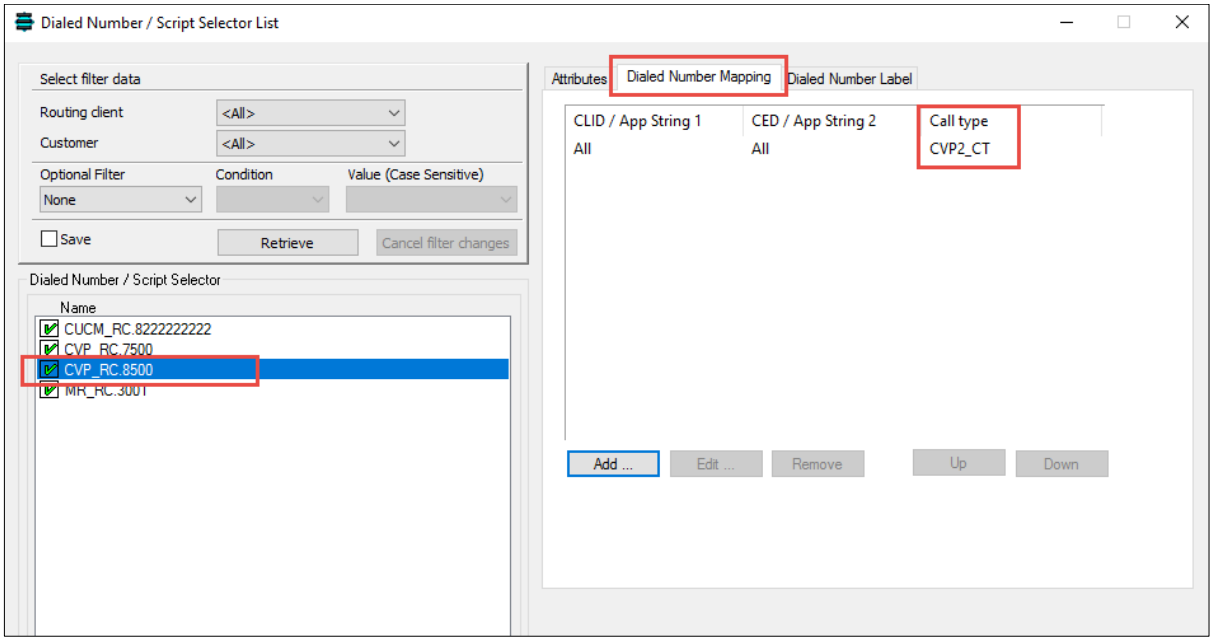

Figure 54 ICM Dialed Number / Script Editor Continuation

5.4.3 Mapping the ICM Script to Dialed Number

The Dialed Number configured in ICM is invoked based on the new call request from CVP. The new script designed needs to be mapped with the required Dialed Number.

- Open **Cisco Unified CCE Tools** and navigate to **Administration tools** and open **Script Editor**
- Navigate to **Script** > **Call_Type Manager**

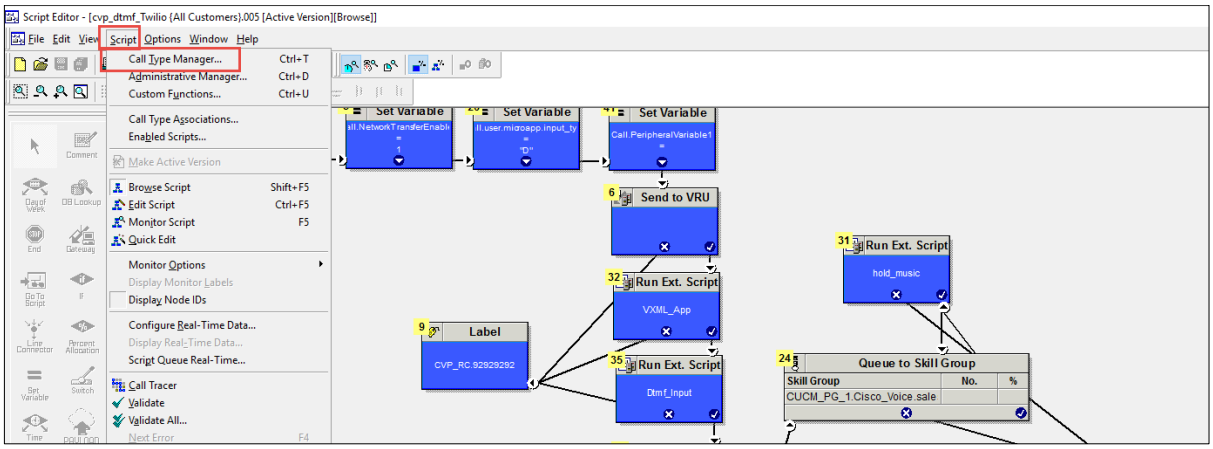

Figure 55 ICM Script mapping with Dialed Number

- In the **Call Type Manager** window, select the **Call Directory** and choose the required **Dialed Number**. In this configuration it is CVP_RC.8500

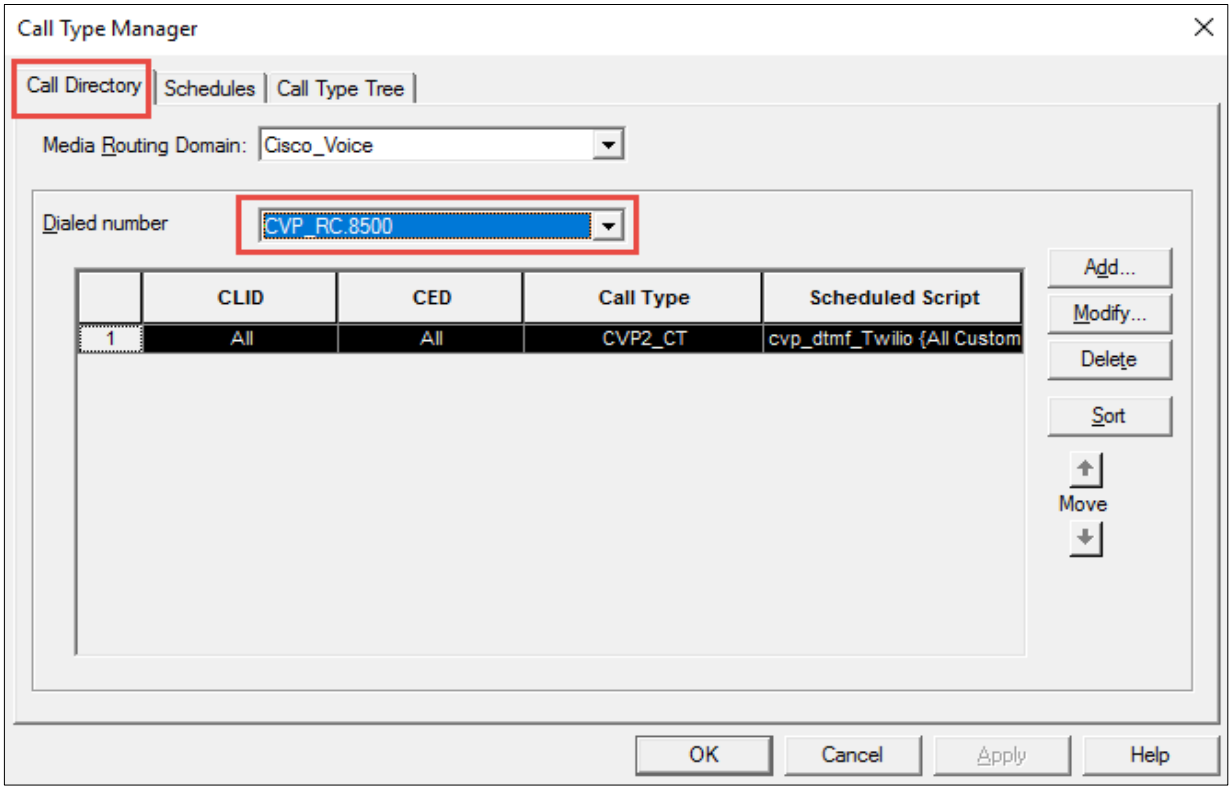

Figure 56 ICM Script mapping with Dialed Number Continuation

- Select Modify and set the **Call Type** for this Dialed Number. In this configuration it is CVP2_CT.

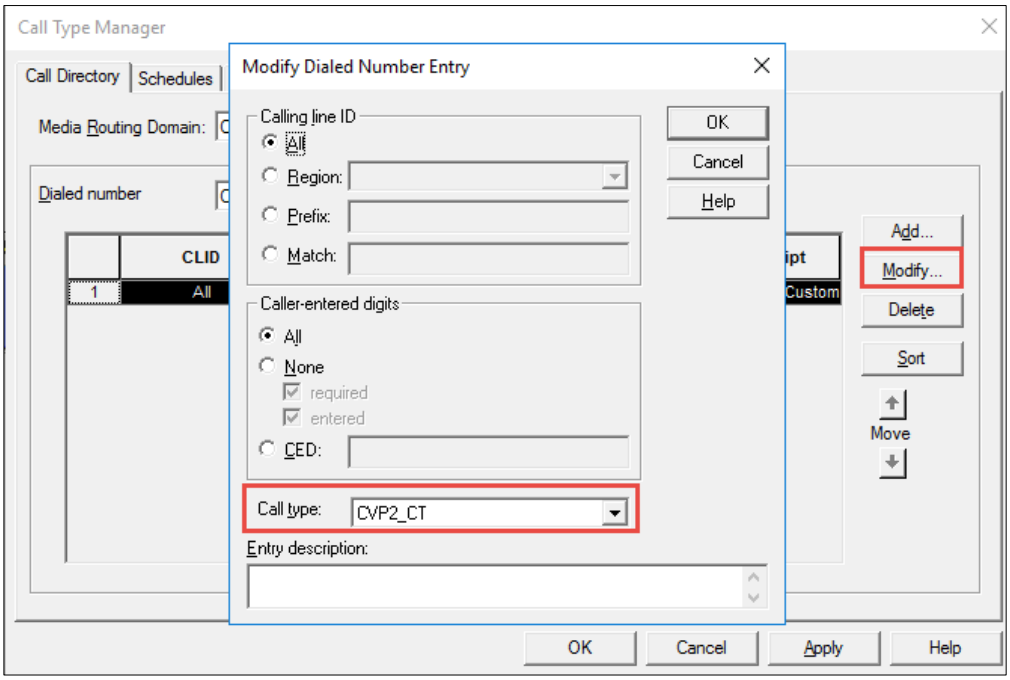

Figure 57 ICM Script mapping with Dialed Number Continuation

- In the **Call Type Manager** window, select **Schedules** and choose the **Call Type**. Click **Modify** to choose the required Call Script. Here cvp_dtmf_Twilio named script is used. Click Apply and OK to save the configuration.

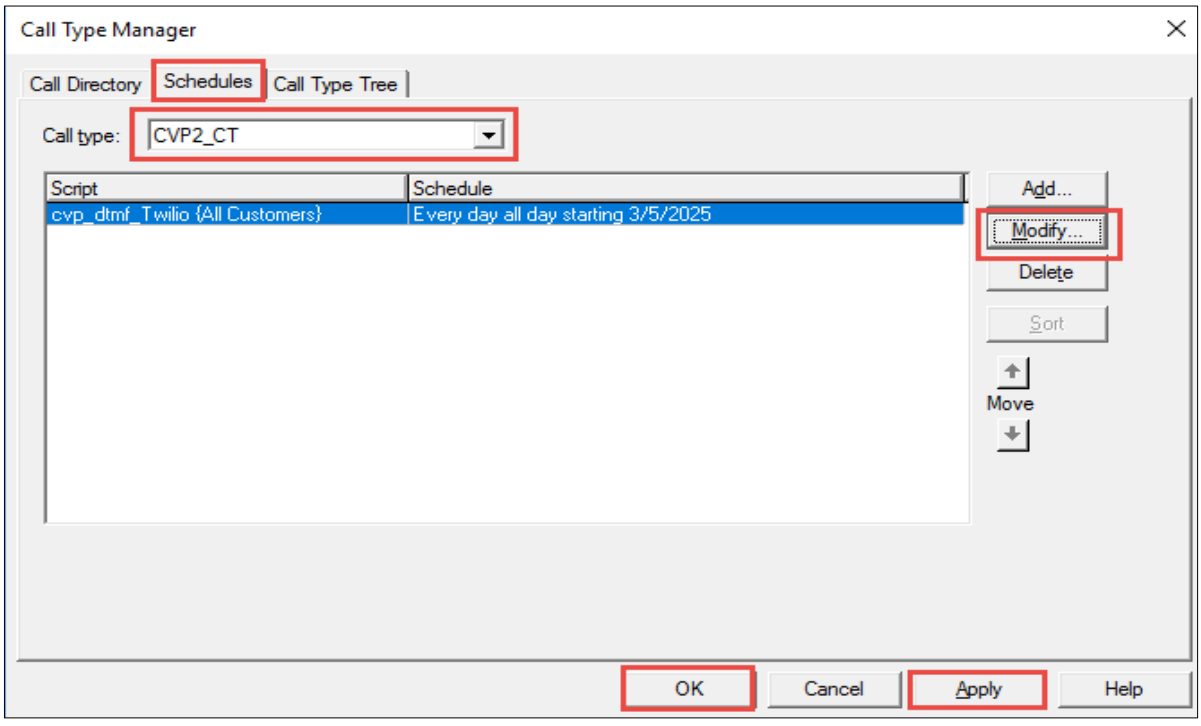

Figure 58 ICM Script mapping with Dialed Number Continuation

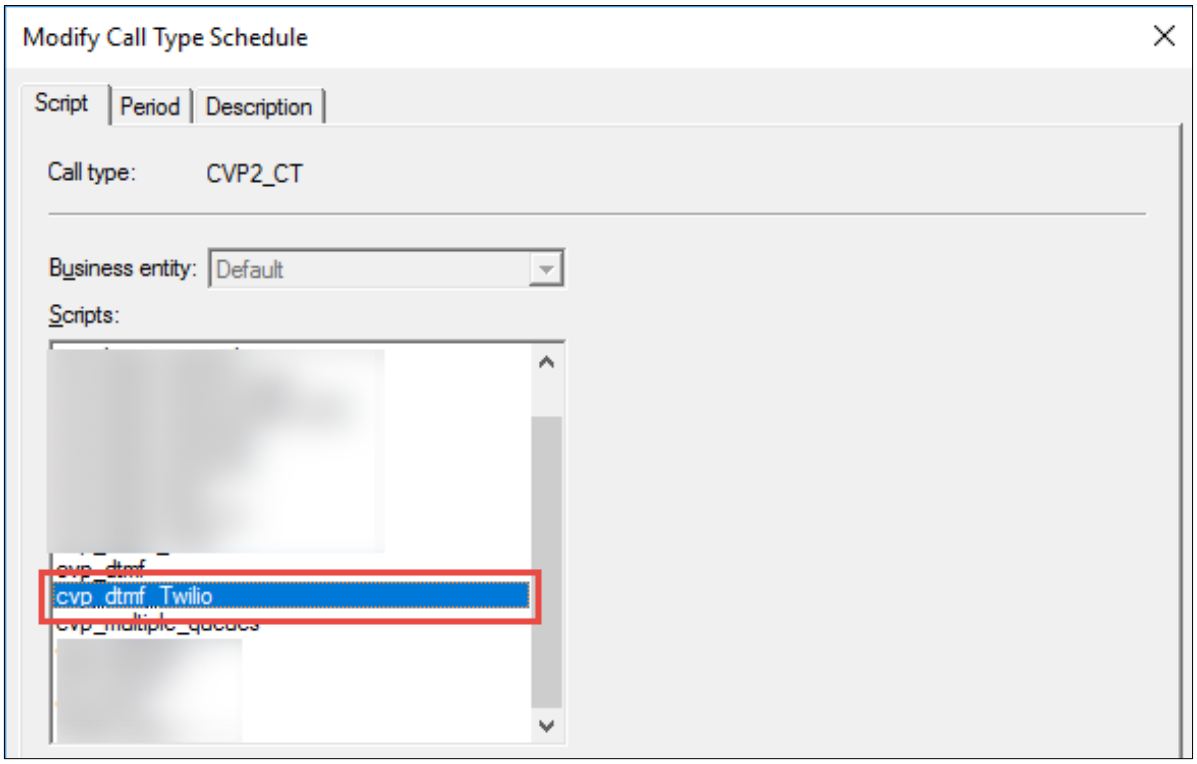

Figure 59 ICM Script mapping with Dialed Number Continuation

5.5 Cisco UCM Device Configuration

Configuration of UCCE Agent Phone is covered in this section. A Cisco desk phone is configured in the UCCE CUCM and is enabled as the associated device for the Finesse Agent.

5.5.1 Configure Agent Phone in CUCM

- Login to CUCM using the administrator credentials to add the phone device.

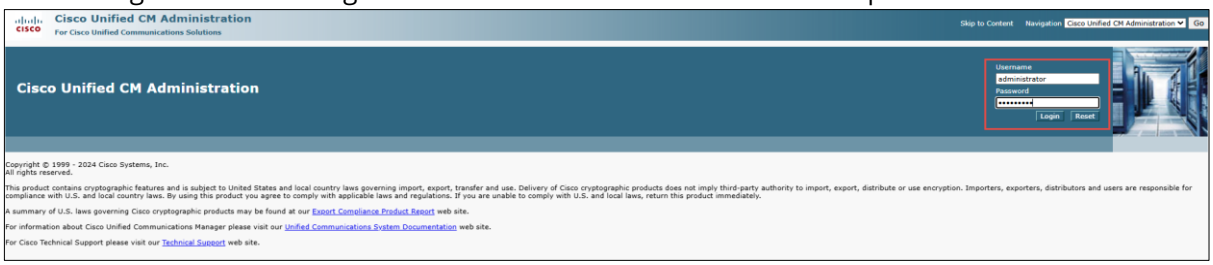

Figure 60 CUCM Login

- Navigate to **Device** > **Phone**
- Select **Add New** to add a new Phone to CUCM by providing the Phone Type. In this configuration it is Cisco 7841

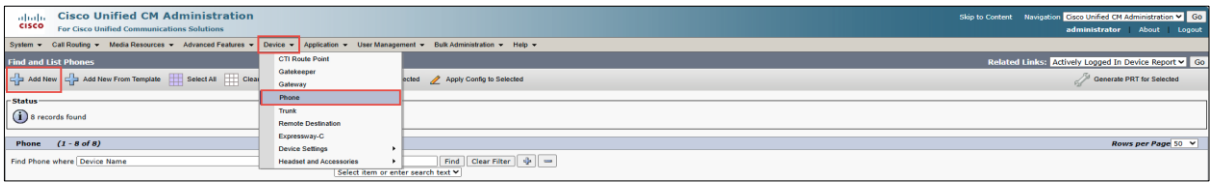

Figure 61 Adding Phone

- Provide the MAC address and complete the Device Information with Device pool, Phone Button template and Softkey Template
- Also provide the details for Media Resource Group list and User Hold and Network Hold MOH audio source

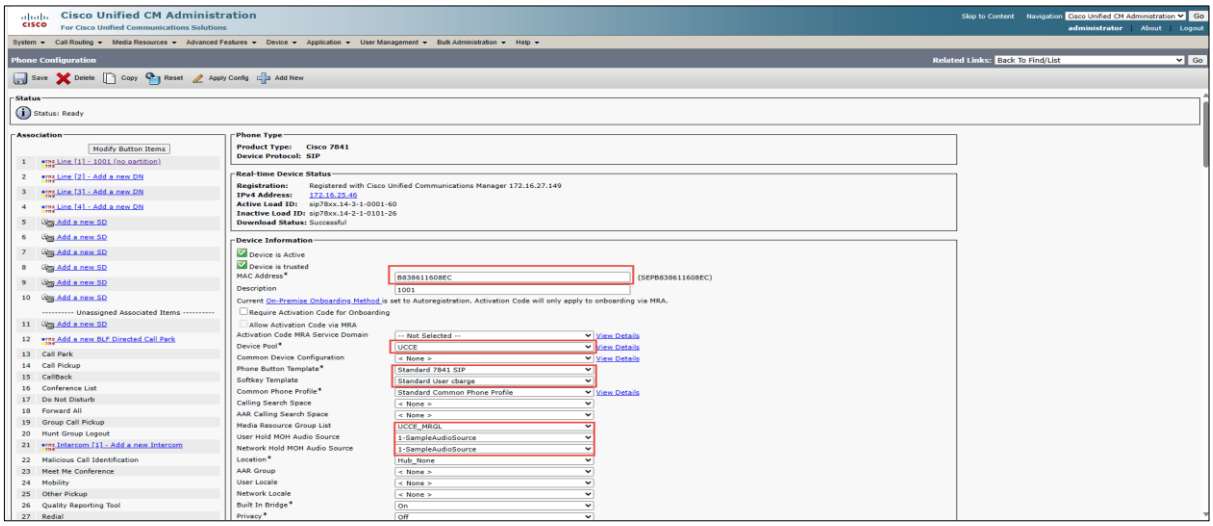

Figure 62 Adding Phone Continuation

- Under Protocol Specific Information select the Device Security Profile with Standard SIP Non-Secure profile of the phone model. Also select a SIP Profile.
- Keep other configurations in this page default and save the configuration.

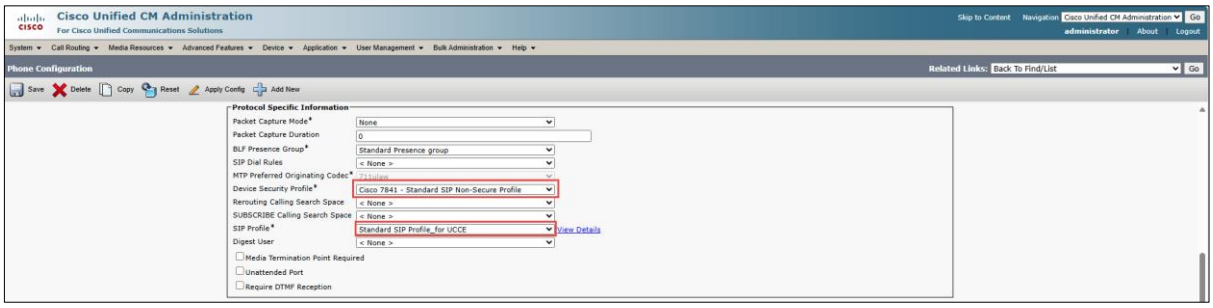

Figure 63 Adding Phone Continuation

- Add a new Line for this Phone by choosing Add a new DN from the left pane
- Provide a Directory Number and Keep the other configurations default.
- Save and Apply Config.

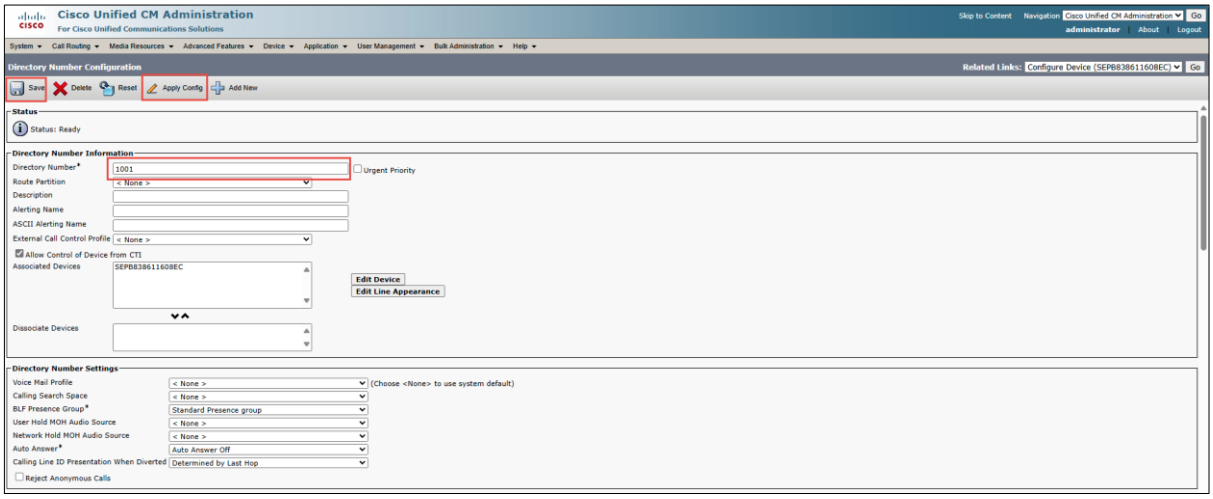

Figure 64 Adding Directory Number

5.5.2 Add Phone to UCCE Application User

- Navigate to **User Management** > **Application User**
- Select the Application User which is related to the UCCE configuration. (UCCE CUCMs will have this Application user created as a part of setup configuration)
- Add the new Phone device to Controlled Devices list using the Device Association.
- Save the configuration.

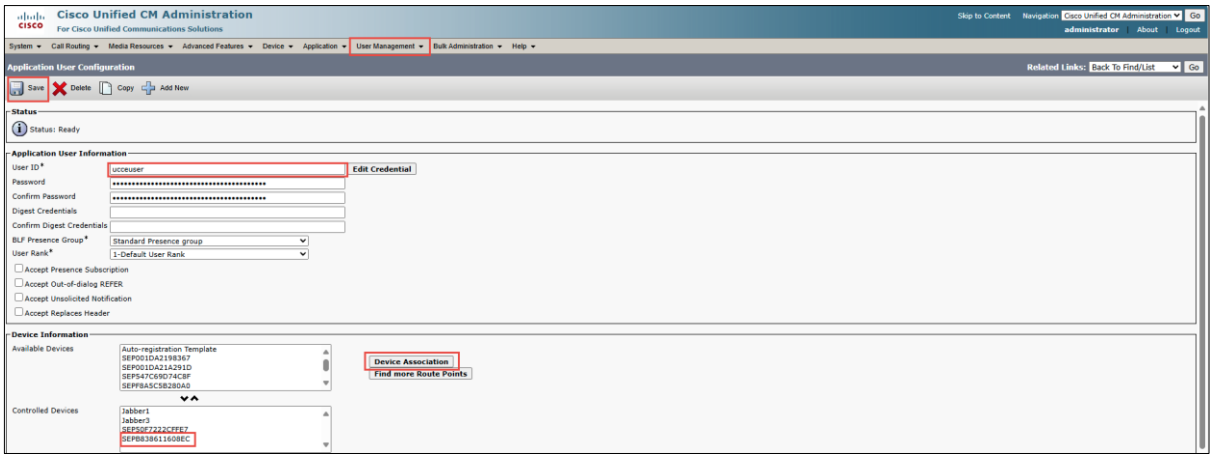

Figure 65 Adding Phone to UCCE Application User

5.5.3 Add Phone to End User

- Navigate to **User Management** > **End User**
- Add a new End User for the Agent Phone

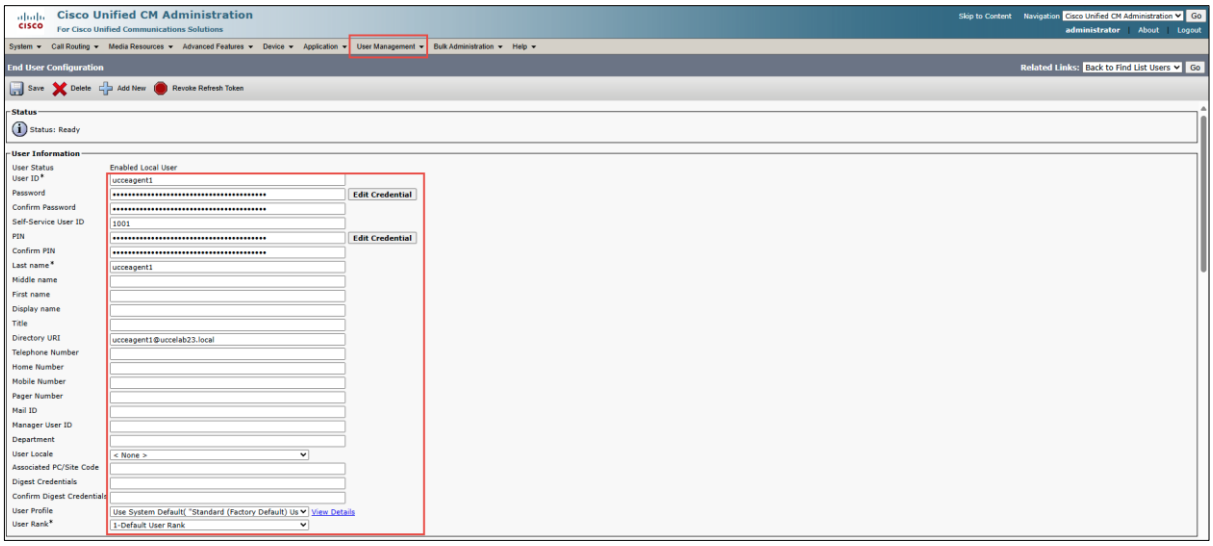

Figure 66 Adding End User

- Add the new Phone device to the End Users Controlled Devices list using Device Association

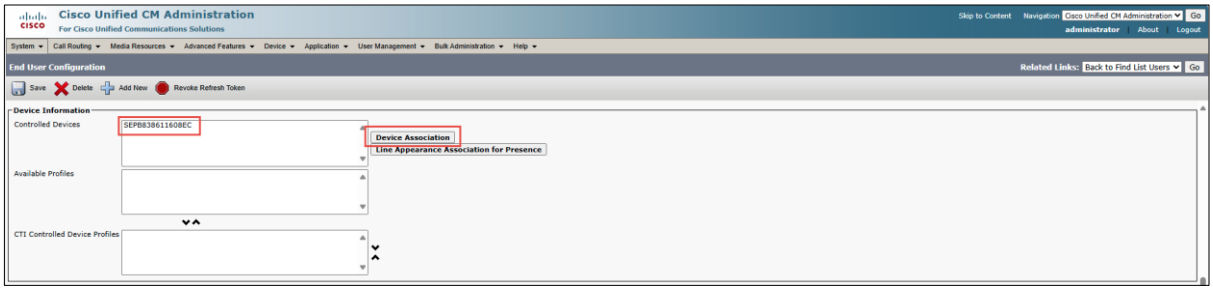

Figure 67 Adding Phone to End User

- Add the required permissions for the End User and Save the configuration.

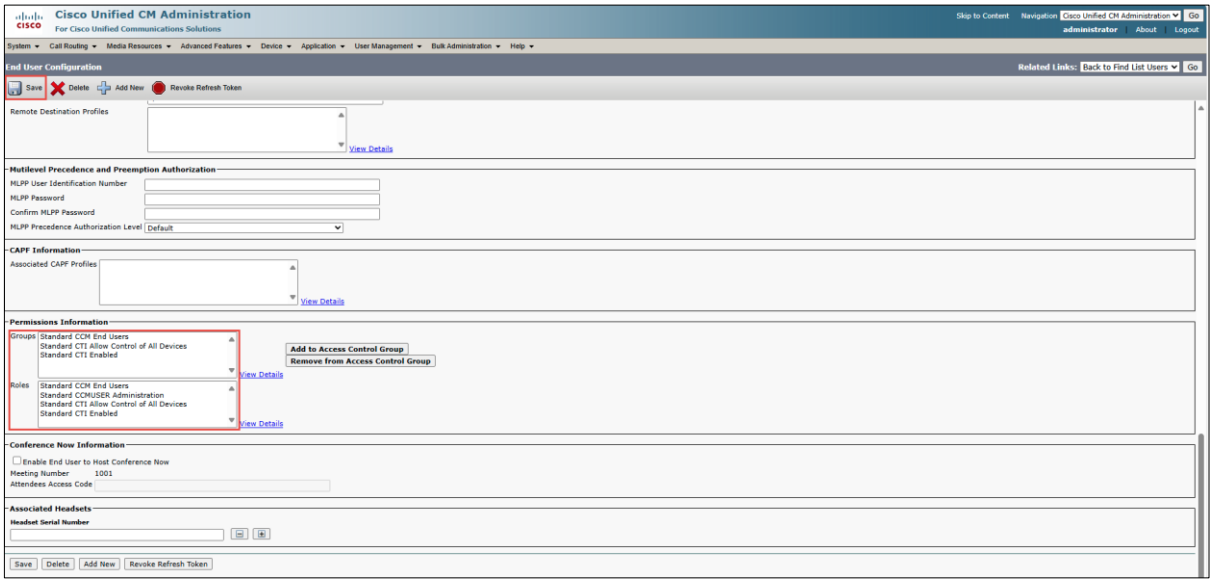

Figure 68 Adding permissions to End User

- Navigate to the newly added Phone Device and set the Owner configuration to the End User created.

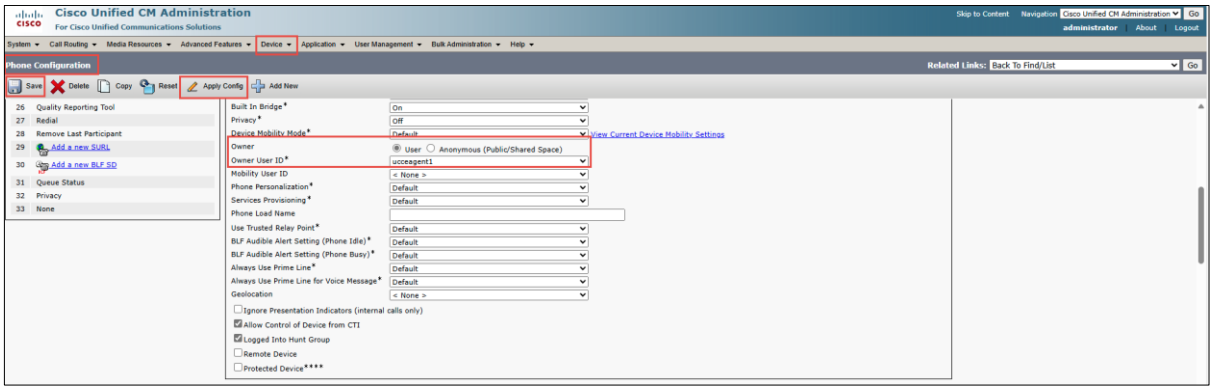

Figure 69 Mapping the End User to the Phone Device

5.6 Cisco Finesse Configuration

Cisco Finesse is configured to have a Screen Pop in the Finesse web client to display the Twilio call context details to the Agent while answering the calls. Screen pop sample gadget for Finesse is used for this configuration.

5.6.1 Setup Finesse Screen Pop Gadget Files

- Download the Screen pop sample gadget for Finesse from Cisco DevNet. (https://developer.cisco.com/docs/finesse/sample-gadgets/#sample-gadgets)
- The downloaded zip file has a folder named “ScreenPop” which contains the required CSS Source File, JSFile and HTML Document

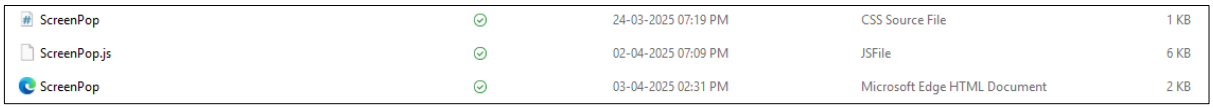

Figure 70 Sample Gadget Files

- Modify the ScreenPop.js file to add the API Get request URL provided by Twilio for retrieving the Call Context details. The URL used here is "**https://cx-data-exchange-7367.twil.io/read_sync?callSid=' + callvars["callVariable1"] + '"**

```
render = function () {
    var currentState = user.getState();
    // html is initially a div tag
    var html = '<div>';

    // for debugging you could print out the agent state and the number of calls (Dialogs)
    //html += '<div id="agentstate"> The current state is: ' + user.getState() + '</div>';
    //html += '<div id="dialogcount"> The number of dialogs is: ' + numDialogs + '</div>';

    if (numDialogs==1) {
        // if we were triggered by a new call (numDialogs==1) then set the html to the url we want to pop in the iframe
        // build the url by adding the callvariable 1 into the search parameter
        html += '<iframe src="https://cx-data-exchange-7367.twil.io/read_sync?callSid=' + callvars["callVariable1"] + '" width="100%" height="650"> </iframe>';

        // comment out the above line and uncomment the line below to change the search engine to bing
        // Note: google search won't allow an iframe, yahoo search has errors too
        //html += '<iframe src="https://www.bing.com/search?q=' + callvars["callVariable1"] + '" width="100%" height="650"> </iframe>';

        // add the closing </div> html element
        html += '</div>';

        clientLogs.log("render(): HTML is: " + html);  // for debugging

        //set the html document's agentout element (see the html after the /script tag) to the html we want to render
        $('#agentout').html(html);

        // automatically adjust the height of the gadget to show the html
        gadgets.window.adjustHeight();
    } else {
        // we don't have a call yet
        html += 'Screen Pop Goes here';
        html += '</div>';

        //set the html document's agentout element to the html we want to render
        $('#agentout').html(html);

        // automatically adjust the height of the gadget to show the html
        gadgets.window.adjustHeight();
    }
},
```

Figure 71 Add API URL in ScreenPop.js file

- Save ScreenPop.js file

- Edit the ScreenPop HTML file to add a Title for the new Screen

```
<Module>
  <ModulePrefs title="TWILIO CALL SUMMARY" description="ScreenPop Gadget">
    <Require feature="settitle"/>
    <Require feature="dynamic-height"/>
    <Require feature="pubsub-2"/>
    <Require feature="setprefs"/>
    <Require feature="loadingindicator">
      <Param name="manual-dismiss">false</Param>
      <Param name="loading-timeout">10</Param>
    </Require>
  </ModulePrefs>
  <Content type="html">
    <![CDATA[ <!DOCTYPE html> <!-- Styling --> <link rel="stylesheet" href="ScreenPop.css" type="text/css" /> <!-- jQuery --> <script
    type="text/javascript" src="/desktop/assets/js/jquery.min.js"></script> <!-- Finesse Library --> <script type="text/javascript"
    src="/desktop/assets/js/finesse.js"></script> <!-- Gadget Business Logic --> <script type="text/javascript" src="ScreenPop.js">
    </script> <body class="claro"> <!-- sample gadget html only has 1 div which will be modified during the screenpop --> <div> <div
    id="agentout"> </div> </body> <script type="text/javascript"> // initialize the gadget running the init handler defined in screenpop.js
    gadgets.HubSettings.onConnect = function () { finesse.modules.SampleGadget.init(); }; </script> ]]>
  </Content>
</Module>
```

Figure 72 Adding Title in HTML file

- Files are ready to upload into the Finesse Server.

5.6.2 Upload Gadget Files to Finesse Server

Finesse allows to upload gadgets and is done via a specific user called "3rdpartygadget". This account only has permission to /files directory and any directories created under it.

- Login to Finesse Server CLI with Username and Password
- Reset the password for 3rdpartygadget using the command "**utils rest_3rdpartygadget_password"**

```
admin:utils reset_3rdpartygadget_password
New Password:
Confirm New Password:
Updating password for 3rdpartygadget...

Password updated successfully.
admin:
```

Figure 73 Password Rest for 3rdpartygadget user

- Use an SFTP application to connect to the Finesse Server. Here **WinSCP** is used.
- Connect to the Finesse Server using **3rdpartygadget** as the Username and the password that is set in the previous step.

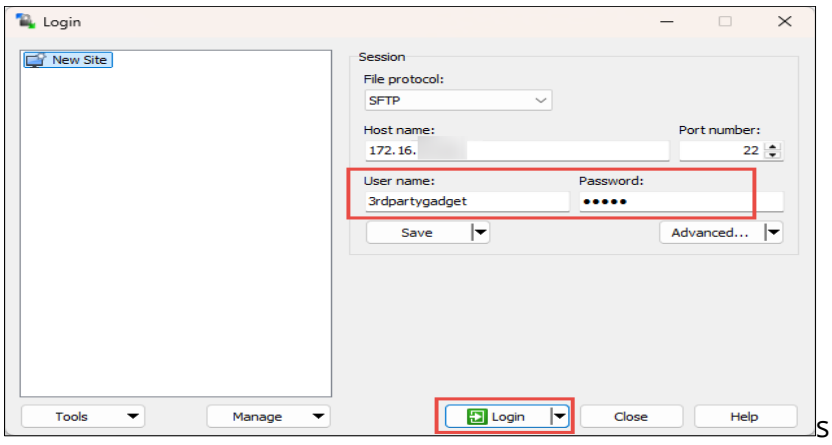

Figure 74 Connect to Finesse Server

- After connecting to Finesse Server, navigate to folder "**files**"
- Copy the CSS Source File, JSFile and HTML Document which is configured in section 5.6.1

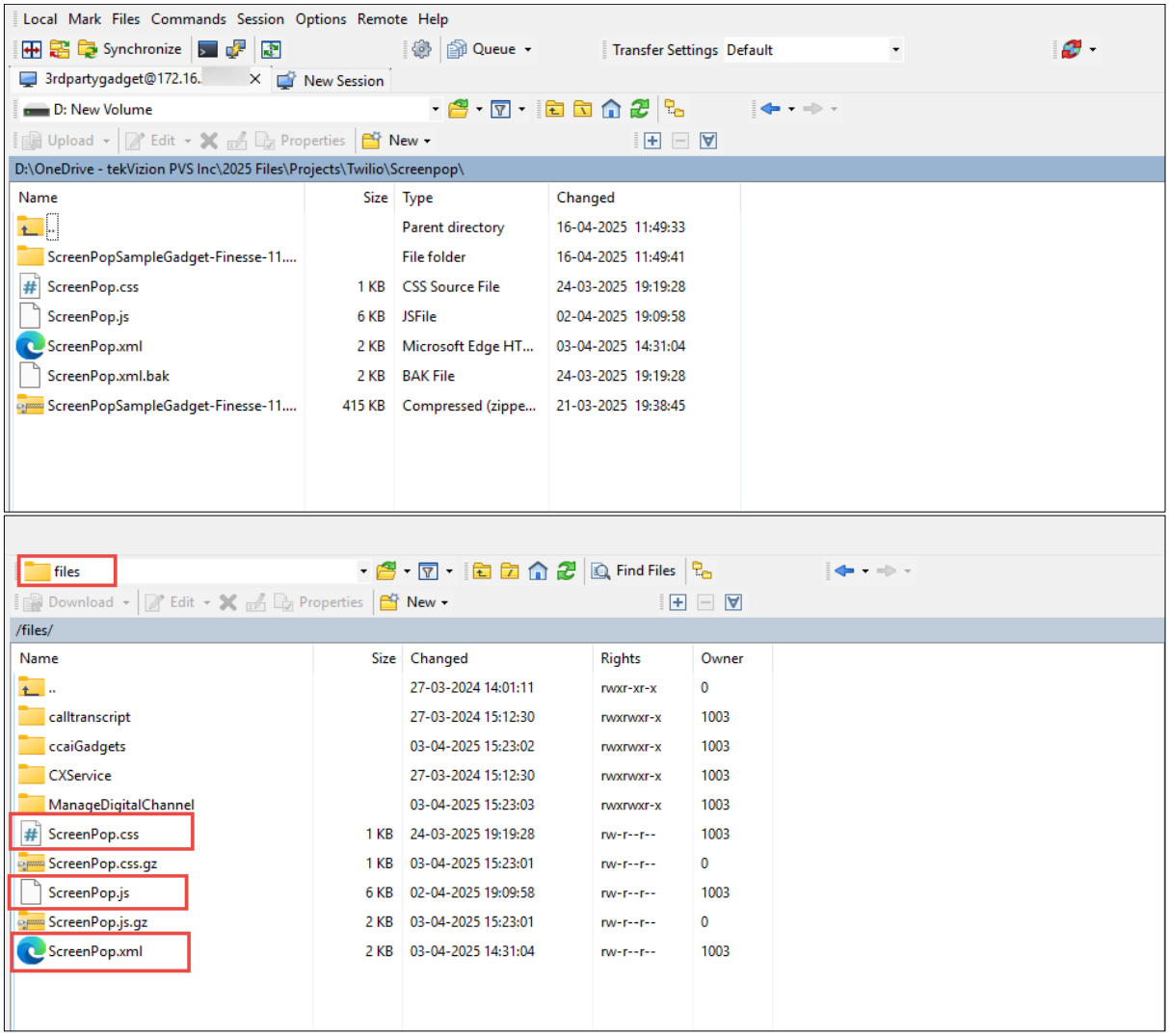

Figure 75 Copying Gadget files to Finesse Server

- All the required files for the ScreenPop gadget are loaded into Finesse Server.

5.6.3 Configure Finesse Desktop Layout

The Finesse Desktop Layout needs to be modified with the new gadget for Screen Pop.

- Login to Finesse Administrator with Username and Password

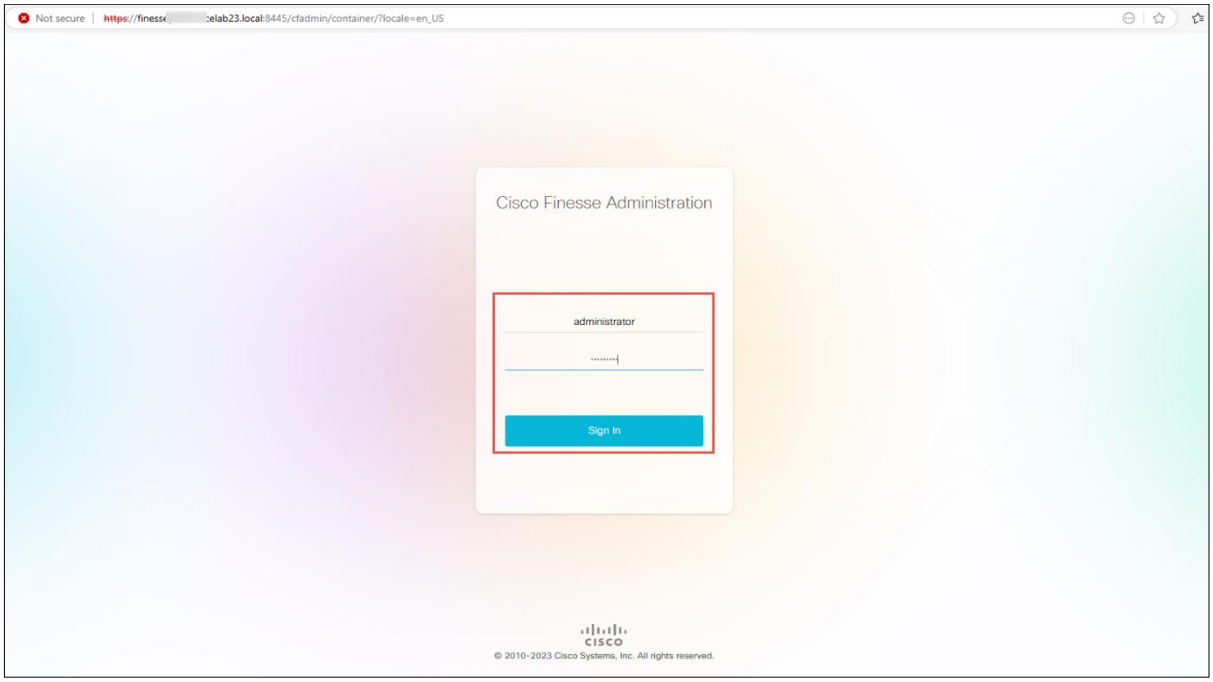

Figure 76 Login to Finesse Administrator

- Navigate to **Desktop Layout** and click **Expand All**

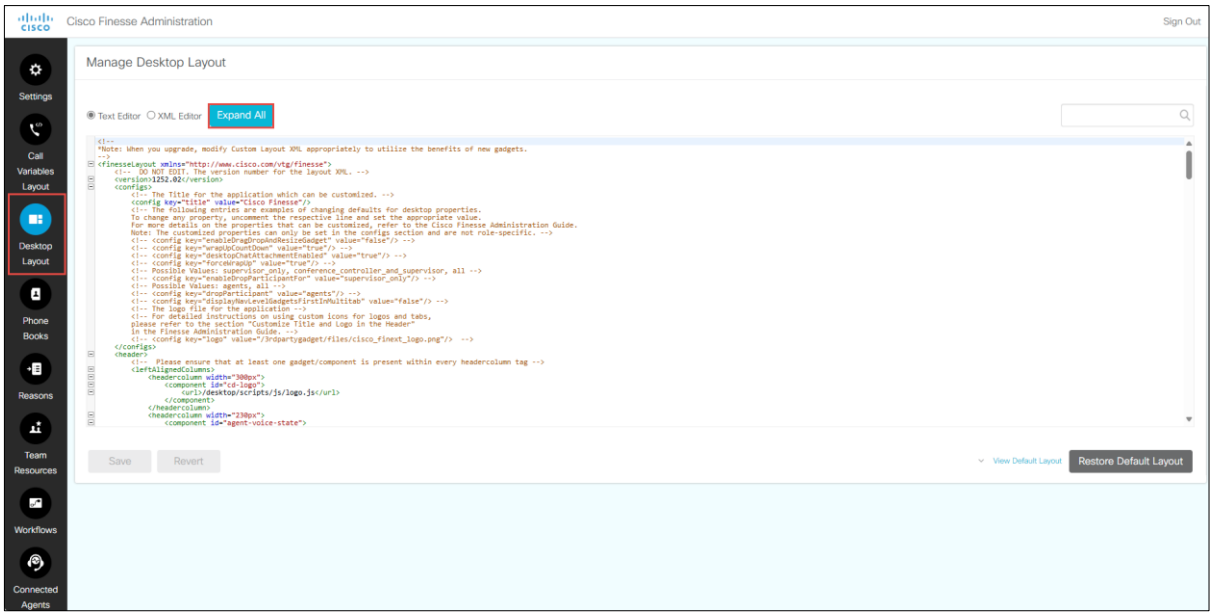

Figure 77 Finesse Desktop

- Locate the Tab for Home page and add the gadget configuration for Screen Pop using below configuration. The ScreenPop.xml file is accessed from the Finesse Server location which is added in section 5.6.2

- Save the Desktop Layout

```
<gadgets>
        <gadget>/3rdpartygadget/files/ScreenPop.xml</gadget>
</gadgets>
```

```
</page>
<tabs>
    <tab>
        <id>home</id>
        <icon>home</icon>
        <label>finesse.container.tabs.agent.homeLabel</label>
        <columns>
            <column>
                <gadgets>
                    <gadget default="true" managedBy="agentMultiTabGadgetContainer">/desktop/scripts/js/queueStatistics.js</gadget>

                    <!-- The Multi-Tab gadget to show live data reports. Uncomment this for displaying the gadgets managed by it. -->
                    <!-- <gadget id="agentReportsMultiTabContainer">/desktop/scripts/js/tabbedGadgets.js</gadget> -->
                    <!--
                    The following Gadgets are for LiveData.
                    If you wish to show LiveData Reports, then do the following:
                         1) Uncomment each Gadget you wish to show.
                         2) Replace all instances of "my-cuic-server.com" with the Fully Qualified Domain Name of your Intelligence Center Server.
                         3) [OPTIONAL] Adjust the height of the gadget by changing the "gadgetHeight" parameter.
                    IMPORTANT NOTES:
                        - In order for these Gadgets to work, you must have performed all documented pre-requisite steps.
                        - Do *NOT* change the viewId (unless you have built a custom report and know what you are doing).
                        - The "teamName" will be automatically replaced with the Team Name of the User logged into Finesse (for Team-specific layouts).  -->

                </gadgets>
                    <gadgets>
                            <gadget>/3rdpartygadget/files/ScreenPop.xml</gadget>
                    </gadgets>
            </column>
        </columns>
    </tab>
```

Figure 78 Adding Screen Pop Gadget to Finesse Desktop

- Navigate to Team Resources in Finesse Administration
- Select Default from the List
- Check the Override the System Default for Desktop Layout Configuration
- Save the configuration.

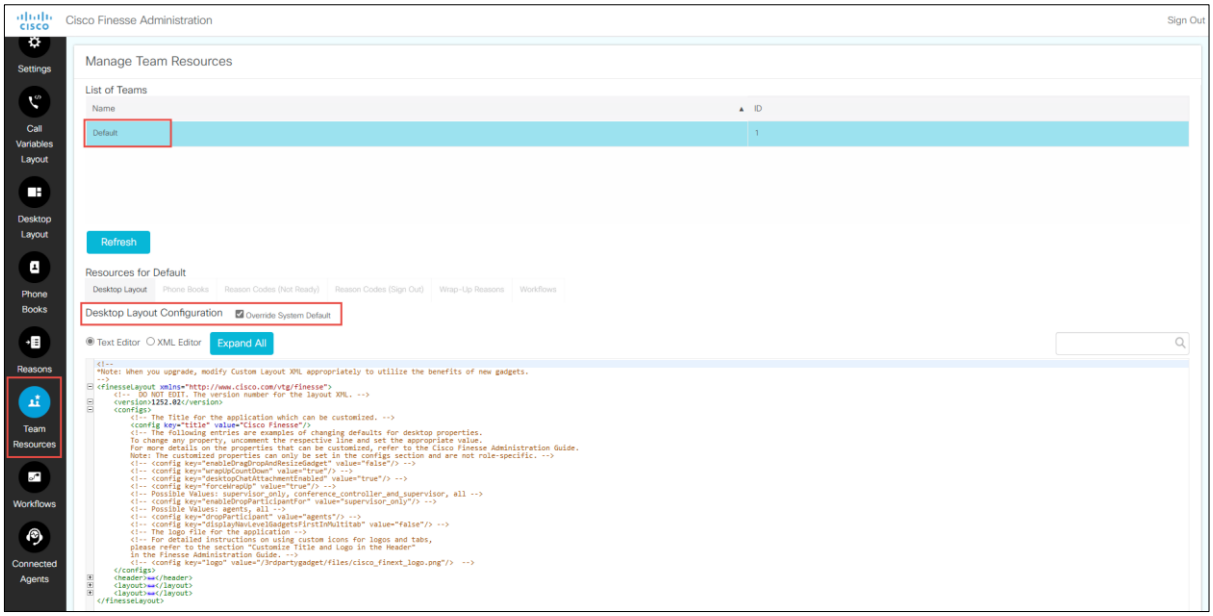

Figure 79 Override System Default

- Login to Finesse CLI and restart Cisco Finesse Tomcat Service and Cisco Tomcat Service using below comments:
    - utils service restart Cisco Finesse Tomcat
    - utils service restart Cisco Tomcat

- Login to Finesse Agent. The new Screen Pop gadget will be available in the Home page as shown below:

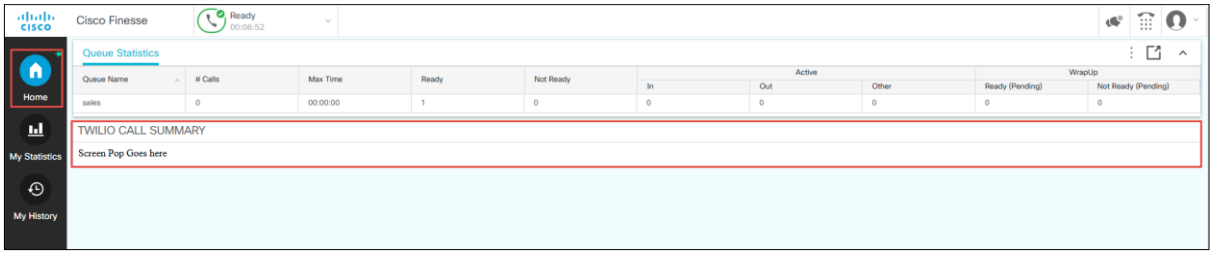

Figure 80 ScreenPop Gadget in Finesse Agent

5.7 Twilio Call Context in Cisco Finesse Agent

The call context captured in Twilio Studio Flow is sent as a response to Finesse Agent's API query and is displayed in the Finesse Agent's Screen Pop section.

5.7.1 Twilio Call Context

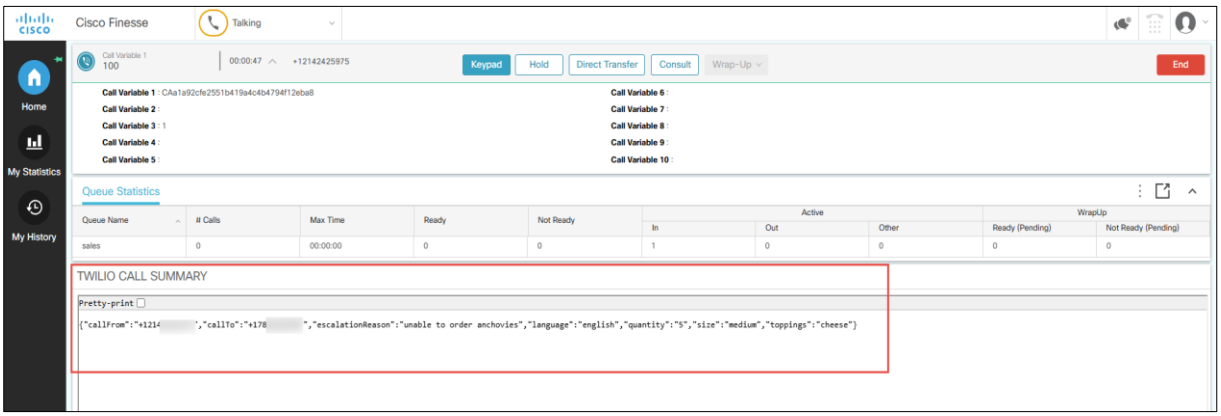

Figure 81 ScreenPop Gadget with Twilio Call Context

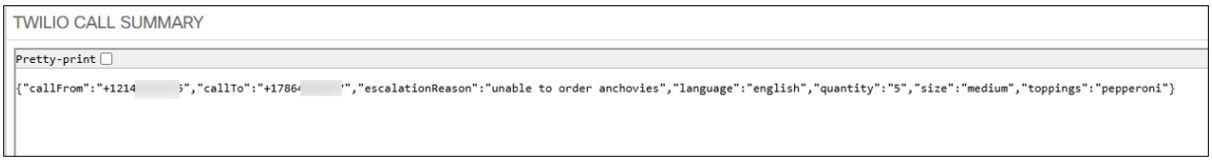

Figure 82 Twilio Call Summary

5.7.2 Call Scenario Tested

- PSTN user calls Twilio–owned number and gets an initial IVR, followed by interaction with a Dialogflow Virtual Agent.
- PSTN user says "Agent" option to reach Cisco UCCE
- Call is connected with UCCE and gets a welcome message
- UCCE provides a selection for required Skill Group
- Based on the selection, call is routed to the available Agent
- Agent answers the call and can view the Twilio Call Context in the Finesse Agent as a ScreenPop

6 Glossary

ACD	Automatic Call Distribution
API	Application Program Interface
CCE	Contact Center Enterprise
CTI	Computer Telephony Integration
CUBE	Cisco Unified Border Element
CUCM	Cisco Unified Communications Manager
CVP	Customer Voice Portal
DC	Domain Controller
DNS	Domain Name System
HTML	Hyper Text Markup Language
ICM	Intelligent Contact Management
IVR	Interactive voice Response
LAN	Local Area Network
OAMP	Operations Console
PG	Peripheral Gateway
PSTN	Public Switched Telephone Network
SFTP	Secure File Transfer Protocol
SIP	Session Initiation Protocol
SSH	Secure Socket Shell
UCCE	Unified Contact Center Enterprise
VRU	Voice Response Unit
VVB	Virtualized Voice Browser
WAN	Wide Area Network

--- End of Document ---