



Zendesk Search for Connect with the help of the Zendesk Support API

Installation and User Guide

Version 2.2 [Mar 2021]



Table of Contents

1	Introduction	3
1.1	Search Parameters	3
1.2	Return Status	4
1.3	Common Fields	5
1.3.1	Common user fields	5
1.3.2	Common ticket fields	5
1.4	Search by Custom Fields	6
1.5	Templates	7
1.5.1	Template string (search_template) and placeholder values	7
1.5.2	Ordering parameters sorty_by and sort_order	7
1.5.3	List of return fields (return_fields)	7
1.5.4	The carry-on (carry_on) facility	8
2	Installation guide	10
3	Using the sample contact flow	16

1 Introduction

Zendesk Search for Amazon Connect is an optional add-on to the [Amazon Connect app for Zendesk](#). It enables driving the business logic of either DTMF driven (classic IVR) or conversation driven (LEX bot) contact flows, based on query results from the Zendesk Support API.

The following types of searches are supported:

- search for user by user ID
- search for user by caller's phone number (CLI)
- search for user by custom Zendesk user fields
- search for ticket by ticket ID
- search for most recent open ticket of an identified user
- templated searches for other queries supported by the Zendesk API

The search is performed within a contact flow by calling a lambda function which is installed as part of the corresponding serverless application within the AWS Serverless Application Repository. This lambda in turn calls the Zendesk Support API with a specific search query, based on parameters passed from the contact flow.

Lambda functions called from Amazon Connect follow a simple interface - accepting a key/value map of arguments and returning another key/value map as a response. In the contact flow, values are then extracted from the map as contact flow attributes of type *External* and using the key as the attribute name or directly as `$.External.<key>` (see [this AWS documentation](#) for more in-depth information).

1.1 Search Parameters

To tell the lambda which of the above search types to use, we pass the parameter `search_by` with one of the following values `zendesk_user`, `calling_user`, `custom_field`, `zendesk_ticket`, `most_recent_ticket` or `template`, followed by none, one or more additional parameters:

search_by type	Additional parameter(s)	Comments
zendesk_user	zendesk_user: Zendesk user ID	Looks for exact match
calling_user	None	Will use detected calling number (CLI) and return the user that has this number as a direct line (not shared)
custom_field	<custom field name>: <value>	Looks for a Zendesk user matching the custom field value. (See an example below)
zendesk_ticket	zendesk_ticket: Zendesk ticket ID	Looks for exact match
most_recent_ticket	zendesk_user: Zendesk user ID, recent_hours: time range	The most recent open ticket for the specified user within a given time range is returned. If the recent_hours is omitted it will default to the value entered at installation.
template	search_template: search query with placeholders for parameter values, <parameter x name>: <value x>	The search template is populated with parameter values and then passed onto Zendesk API. (See detailed discussion below)

Below is an example of configuring the lambda call block in a contact flow, where we want to search for a Zendesk support ticket by its number captured in a previous prompt:

Zendesk Search for Connect with the help of the Zendesk Support API

Function input parameters

☒ Use text ✕

Destination key
search_by

Value
zendesk_ticket

☐ Use attribute

☐ Use text ✕

☒ Use attribute

Destination key
zendesk_ticket

Type
System

Attribute
Stored customer input

[Add another parameter](#)

1.2 Return Status

Depending on the search type, the response from lambda will have different fields in the key/value map. All responses will have a status field, which can have one of the following values:

- ok: the search was successful, we have found at least one matching user or ticket
- not found: the search couldn't find any user or ticket based on given parameters
- bad request: some parameters are missing or invalid
- server error: there was a networking error or some other technical issue

Here's a cut-out from a contact flow depicting how we may branch based on the status value returned from the lambda:

1.3 Common Fields

Apart from the status field, each response also contains some common fields, depending on what we're searching for (i.e. a user or a ticket):

1.3.1 Common user fields

Field name	Zendesk API property	Field value
*zendesk_user	id	Unique user ID within the Zendesk system.
*customer_name	name	User's full name as stored in Zendesk
*customer_number	phone	User's primary phone number
external_id	external_id	A unique identifier from another system, eg. accounts
user_name	N/A	User's first name (extracted from full name)
user_email	email	User's email address
user_role	role	User's role. Possible values are end-user, agent, or admin
user_locale	locale	User's locale, eg. en-AU
time_zone	iana_time_zone	User's time zone, eg. Australia/Sydney
organization	organization_id	The id of the user's organisation - see Zendesk documentation for more information
suspended	suspended	Whether the user is suspended

* These value are usually stored in custom attributes with the same name and passed to the Amazon Connect app for Zendesk.

For more details on the Zendesk API user properties please check [this documentation](#).

1.3.2 Common ticket fields

Field name	Zendesk API property	Field value
*zendesk_ticket	id	Unique ticket ID within Zendesk system.
*zendesk_user	requester_id	The ID of the user who requested this ticket
ticket_subject	subject	The value of the subject field for this ticket
ticket_status	status	Current ticket status. Can be new, open, pending, hold, solved, or closed.

Zendesk Search for Connect with the help of the Zendesk Support API

Field name	Zendesk API property	Field value
ticket_brand	brand_id	The ID of the brand this ticket is associated with

* These value are usually stored in custom attributes with the same name and passed to the Zendesk Connect app.

For more details on Zendesk API ticket properties please check [this documentation](#).

This returned fields can then be used to further drive business logic within the contact flow. For example `user_locale` can be used to play prompts in the customer's native language, `brand_id` and `organization` can determine routing to appropriate queue or another contact flow, etc.

1.4 Search by Custom Fields

Within Zendesk we can create our own custom user fields to store various data about our customers. We can then use the name of such custom fields in our search.

Let's say for example that we've added a field to store a customer's account number:

The screenshot shows the 'User Fields' management interface. On the left, under 'Active fields (2)', there is a list of fields: 'Account Number' and 'Subscription Plan'. The 'Account Number' field has 'deactivate' and 'clone' buttons. On the right, a modal is open for editing the 'Account Number' field. The modal contains the following fields: 'Field title shown to agents' (set to 'Account Number'), 'Field key' (set to 'account_number' with a warning that it cannot be changed after creation), and 'Description (optional)' (set to 'Account Number from external finance database'). At the bottom of the modal are 'Delete', 'Cancel', and 'Update field' buttons.

When we added this field we had to specify a unique *Field key*. We will use this field key as the parameter name in our lambda call, like this:

1.5 Templates

Searching with a template provides the greatest power and flexibility but is more complex and requires some upfront preparation. To be productive with this type of search one needs to be familiar with the [Zendesk Support search reference](#).

This type of search requires four groups of parameters to be passed to the lambda:

- *template string* with placeholders, eg. `search_template = type:user email:{user_email}`
- *placeholder values*, eg. `user_email = mary.smith@widgets.biz`
- (optional) *ordering parameters* `sort_by` and `sort_order`
- (optional) *list of return fields*, eg. `details`, `notes`

Let's have a closer look at each of them.

1.5.1 Template string (`search_template`) and placeholder values

A search string, formatted according to the Zendesk search syntax, except that the values are expressed as placeholders - parameter names inside braces, eg. `{zendesk_ticket}`. Before the search string is submitted to the Zendesk Support API, all placeholders are replaced with values of corresponding parameters.

For example, say we pass 83217 as `zendesk_user` and 48 as `recent_hours`, then this template
`type:ticket requester:{zendesk_user} -status:closed -status:solved created>{recent_hours}`
hours

becomes:

`type:ticket requester:83217 -status:closed -status:solved created>48hours`

1.5.2 Ordering parameters `sort_by` and `sort_order`

Searching via templates often returns a list of several items. The lambda function however, can only return a single one back to the contact flow. As a rule, this is the first item in the list. That's why it may be important to sort the result set in a certain way so that the desired item is at the start of the list.

- Parameter `sort_by` contains the name of the property we want the returned results to be ordered by.
- Parameter `sort_order` can be either `asc` or `desc` and specifies either ascending or descending sort.

Following the above template example we would set the `sort_by` parameter to `updated` and `sort_order` to `desc`, so that we get the most recent open ticket as the first item in the result set.

1.5.3 List of return fields (`return_fields`)

This parameter contains a comma separated list of fields from the result set that should be included in the lambda response, in addition to common fields (as mentioned earlier in this document) that are always returned. Common fields set is determined by the `type` search attribute at the start of the search template, so it's always a good idea to include it when searching for either tickets or users.

For example, a search template starting with `type:ticket` will return all the common ticket fields listed above, plus any additional fields listed in the `return_fields` parameter.

All responses from a templated search will also return one more property - `count`. This reflects the number of results found by the search and can be used in driving the contact flow logic.

1.5.4 The carry-on (carry_on) facility

When using Zendesk search within our contact flows we often need to make consecutive lambda calls and sometimes there's a need to keep values returned from one call in the next call's response. In this scenario the carry-on facility comes handy. When doing the second lambda call we can use a parameter called `carry_on` containing a (comma separated) list of fields returned from the first call and then add individual fields as parameters with their values. Then the response from the second call will contain also those fields.

For example, if we first searched for a user and then for the most recent ticket for that user we may want to carry on certain user fields, such as the user's first name in the ticket response:

The image shows two screenshots of the Zendesk Connect configuration interface, illustrating the 'carry-on' facility. The top screenshot shows the 'Use text' option selected, with 'Destination key' set to 'carry_on' and 'Value' set to 'user_name'. The bottom screenshot shows the 'Use attribute' option selected, with 'Destination key' set to 'user_name', 'Type' set to 'External', and 'Attribute' set to 'user_name'.

☒ Use text ✕

Destination key
`carry_on`

Value
`user_name`

☐ Use attribute

☐ Use text ✕

☒ Use attribute

Destination key
`user_name`

Type
External ▼


Attribute
`user_name`

We can then use both a ticket field and the carried over `user_name` field in a prompt:

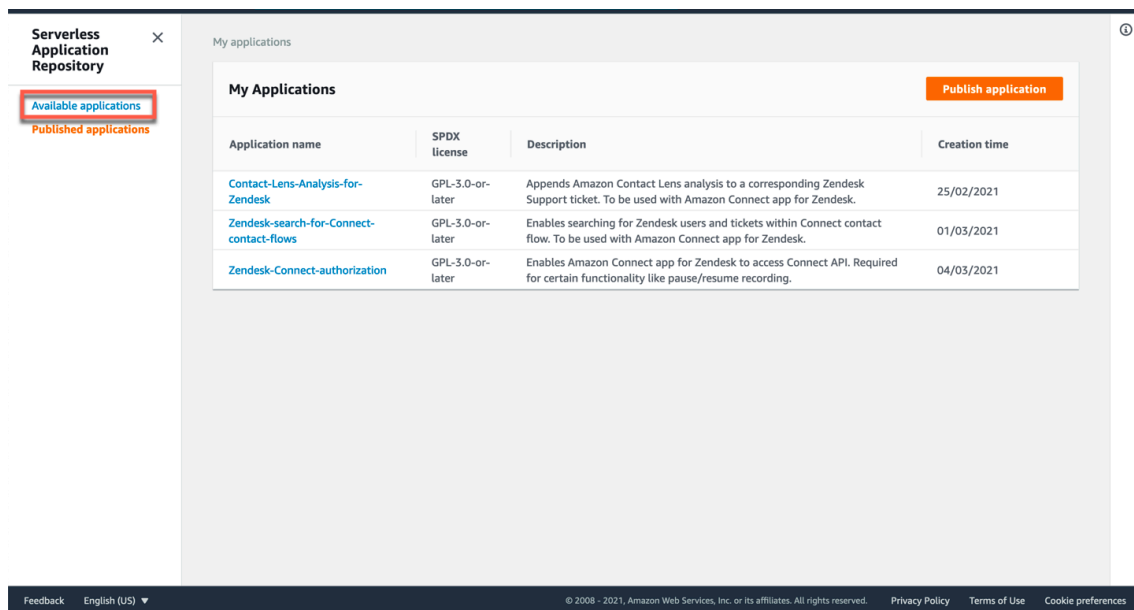
Zendesk Search for Connect with the help of the Zendesk Support API

2 Installation guide

To enable this feature you will need to install a serverless application within the AWS Serverless Application Repository. Sign in to your AWS account, then search for and select the *Serverless Application Repository* service.

 Make sure you are in the same region as your Connect instance.

Click on *Available applications*.



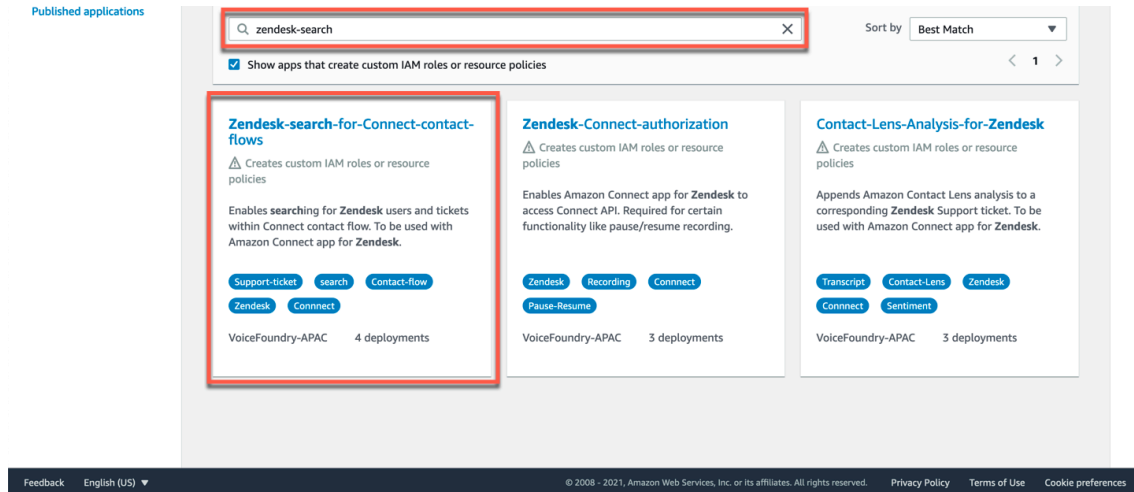
The screenshot shows the AWS Serverless Application Repository console. On the left, the 'Serverless Application Repository' sidebar is visible, with 'Available applications' highlighted in a red box. The main content area is titled 'My applications' and contains a table of applications. The table has four columns: 'Application name', 'SPDX license', 'Description', and 'Creation time'. Three applications are listed:

Application name	SPDX license	Description	Creation time
Contact-Lens-Analysis-for-Zendesk	GPL-3.0-or-later	Appends Amazon Contact Lens analysis to a corresponding Zendesk Support ticket. To be used with Amazon Connect app for Zendesk.	25/02/2021
Zendesk-search-for-Connect-contact-flows	GPL-3.0-or-later	Enables searching for Zendesk users and tickets within Connect contact flow. To be used with Amazon Connect app for Zendesk.	01/03/2021
Zendesk-Connect-authorization	GPL-3.0-or-later	Enables Amazon Connect app for Zendesk to access Connect API. Required for certain functionality like pause/resume recording.	04/03/2021

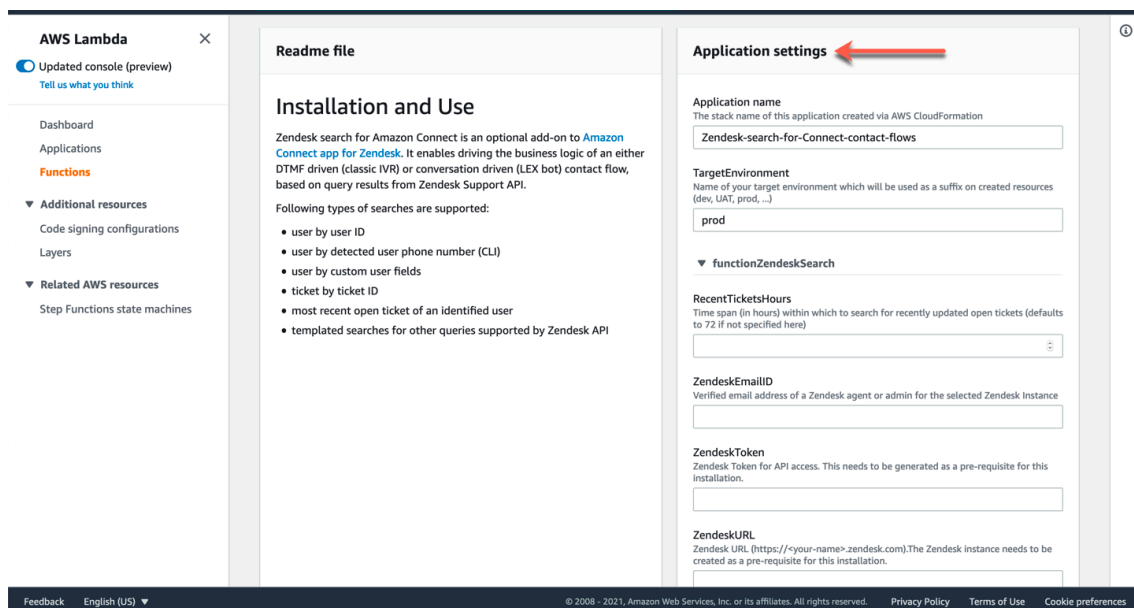
Under *Public applications*, select the checkbox *Show apps that create custom IAM roles or resource policies*.

Zendesk Search for Connect with the help of the Zendesk Support API

Search for and select *Zendesk-search-for-Connect-contact-flows*.



Scroll down to *Application settings* and enter the following information:



Application name

Leave this as is.

TargetEnvironment

This is the name of your target environment (dev, UAT, prod etc). If unsure, just leave as *prod*.

RecentTicketsHours

This applies for one of the search options to look for the most recent ticket to open for a user. Set the time frame in hours within which to search for the most recently updated open ticket for a user (defaults to 72 if not specified here).

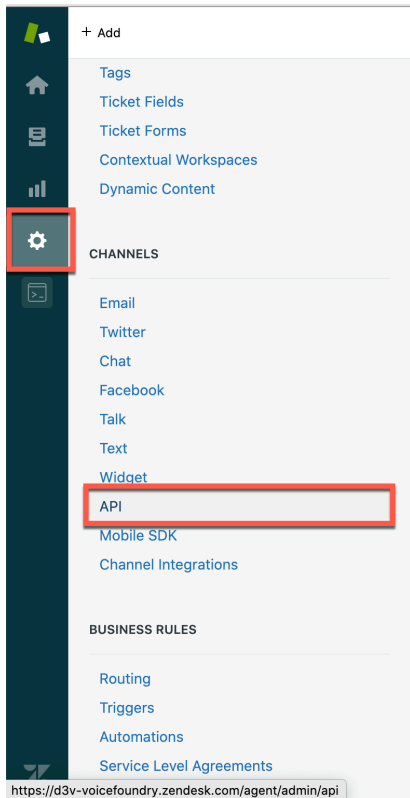
Zendesk Search for Connect with the help of the Zendesk Support API

ZendeskEmailID

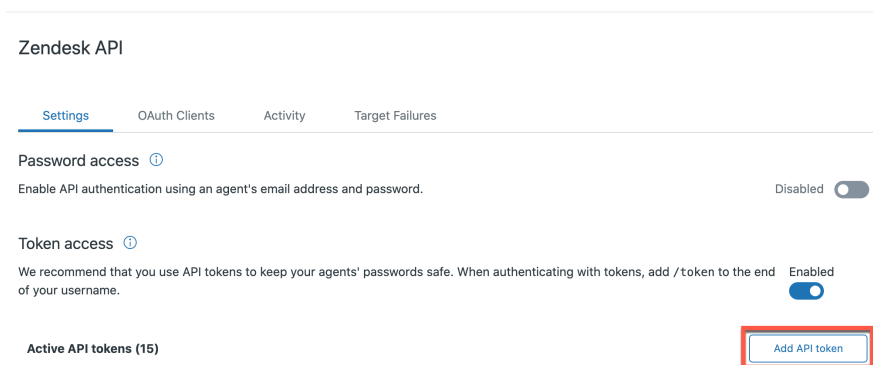
Enter the verified email address of a Zendesk user for your Zendesk instance.

ZendeskToken

Sign in to your Zendesk instance. Click on *Admin* within the left hand navigation bar, and under *Channels* select *API*.



Click on *Add API token*.



Click on *Copy*. Paste the API token into the *ZendeskToken* field.

Zendesk Search for Connect with the help of the Zendesk Support API

ZendeskURL

Enter the URL of your Zendesk instance. Ensure the URL begins with `https://` and do not include `/` at the end of the URL or the stack will fail.

Select the checkbox to *create custom IAM roles* and click on *Deploy*.

The screenshot shows the 'Installation and Use' page for an AWS Lambda application. The left sidebar contains navigation links for 'AWS Lambda', 'Updated console (preview)', 'Dashboard', 'Applications', 'Functions', 'Additional resources', and 'Related AWS resources'. The main content area is titled 'Installation and Use' and includes a description of the application, supported search types, and a list of supported search types. The right sidebar contains configuration fields: 'Application name' (Zendesk-search-for-Connect-contact-flows), 'TargetEnvironment' (prod), 'functionZendeskSearch', 'RecentTicketsHours' (120), 'ZendeskEmailID', 'ZendeskToken', and 'ZendeskURL'. A checkbox labeled 'I acknowledge that this app creates custom IAM roles' is checked and highlighted with a red box. At the bottom right, there are 'Cancel', 'Previous', and 'Deploy' buttons.

Once your stack has been successfully created, scroll down and take note of the lambda name.

The screenshot shows the 'Overview' page for the 'Zendesk-search-for-Connect-contact-flows' application. The left sidebar contains navigation links for 'AWS Lambda', 'Updated console (preview)', 'Dashboard', 'Applications', 'Functions', 'Additional resources', and 'Related AWS resources'. The main content area is titled 'Getting started' and includes a 'Set up your development environment' section with links to Visual Studio, Visual Studio Code, JetBrains, and AWS SAM CLI. Below this is a 'Resources (2)' section with a table listing the application's resources. The table has columns for 'Logical ID', 'Physical ID', 'Type', and 'Last modified'. The first row shows the 'functionZendeskSearch' resource with the physical ID 'zendeskSearchInContactFlow-uat', which is highlighted with a red arrow.


Logical ID	Physical ID	Type	Last modified
functionZendeskSearch	zendeskSearchInContactFlow-uat	Lambda Function	2 days ago

In your AWS account, search for and select Amazon Connect. Click on your Amazon Connect instance.

Zendesk Search for Connect with the help of the Zendesk Support API

Amazon Connect virtual contact center instances

Select a virtual contact center instance to manage its directory, administrator(s), telephony options, data storage, and advanced features.

[Add an instance](#) [Remove](#) 

Instance Alias	Access URL	Channels	Create Date	Status
<input type="checkbox"/> [redacted]	[redacted]	Inbound, outbound telephony	05/05/2020	Active
<input type="checkbox"/> [redacted]	[redacted]	Inbound, outbound telephony	05/05/2020	Active

Click on *Contact flows*.

Amazon Connect > vf-dev-1

Overview

Telephony

Data storage

Data streaming

Analytics tools

Tasks

Customer profiles

Approved origins

Contact flows

Under *AWS lambda*, click on the *Function* dropdown and select the lambda that was created during the stack.

Zendesk Search for Connect with the help of the Zendesk Support API

Click on *Add lambda function*.

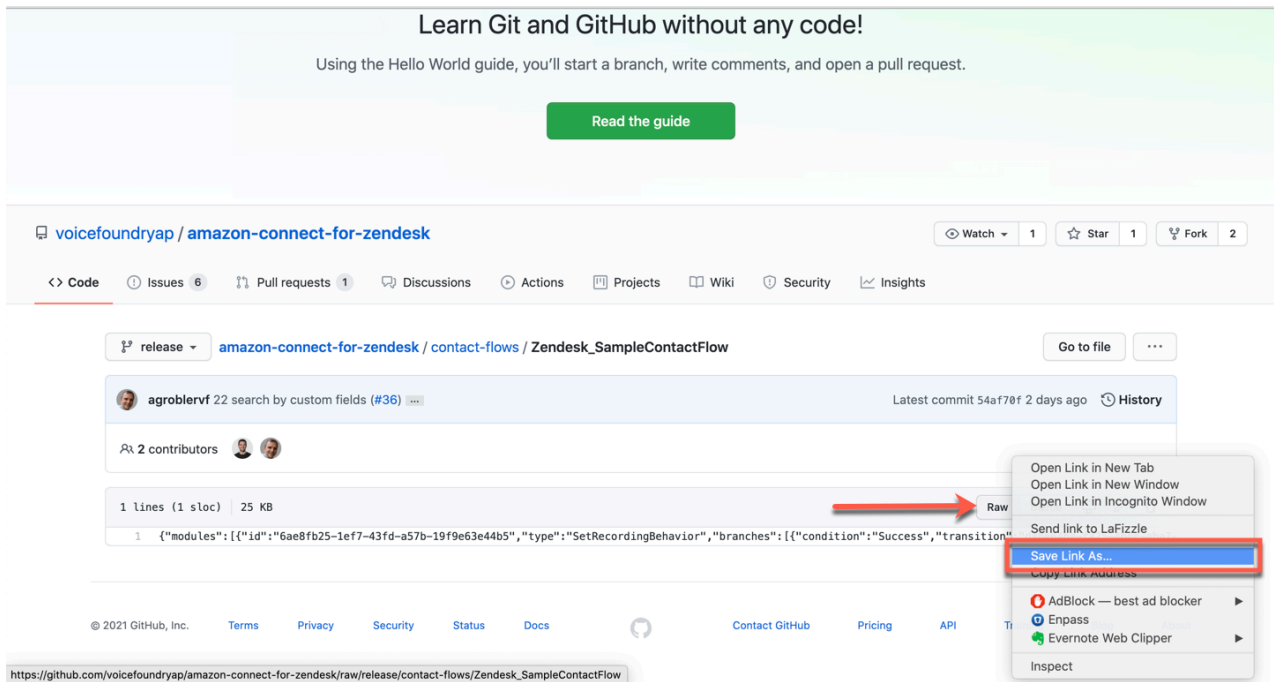
The screenshot shows the AWS Lambda console interface. At the top, there's a section titled 'AWS Lambda' with a brief description of its capabilities. Below this, a note states: 'By adding Lambda functions, you are granting Amazon Connect permission to invoke them [Create a new Lambda function](#)'. A dropdown menu is set to 'zendeskSearchInContactFlow-ust', and a red box highlights the '+ Add Lambda Function' button. Below the button, a table lists existing Lambda functions. At the bottom, there's a 'Contact flow logs' section with a checkbox to 'Enable Contact flow logs' and a note that logs will be stored at '/aws/connect/vf-dev-1'.

Function	Function Name	Remove
ZendeskIntegration-6P15O5KP9WJJ	arn:aws:lambda:ap-southeast-2:214558022353:function:rtrial0129-bayutest-ZendeskIntegration-6P15O5KP9WJJ	Remove
initContactDetails-XYF1M89MG1L	arn:aws:lambda:ap-southeast-2:214558022353:function:rtrial0129-bayutest-initContactDetails-XYF1M89MG1L	Remove
kvsConsumerTrigger-174X4YY2YL009	arn:aws:lambda:ap-southeast-2:214558022353:function:rtrial0129-bayutest-kvsConsumerTrigger-174X4YY2YL009	Remove
zendeskContactInit-dev	arn:aws:lambda:ap-southeast-2:214558022353:function:zendeskContactInit-dev	Remove

Once you have added the lambda to your Connect instance, you will then need to add the lambda to your contact flows. To get an understanding of how to configure your contact flow with this lambda, view the sample contact flow below.

3 Using the sample contact flow

We have provided a sample contact flow that showcases some of the potential searches you can perform within your own contact flow. The sample contact flow is located in our GitHub account, and can be downloaded [here](#). To download the file within our GitHub account, click on *Raw*, then click on *Save Link As* and save the file to your desktop.



NOTE: You will need to update the lambdas in the sample contact flow with the lambda you added to your Connect instance.