Quintrix
A MINDLANCE COMPANY

AWS Implementations

MODULE 3.7 – BATCH IMPLEMENTATIONS

1095 Morris Ave, SUITE 101, Union NJ 07083
646-762-0305 | INFO@QUINTRIXSOLUTIONS.COM

Modernize Batch Implementation

This process was posted on the AWS Partner Network (APN) by Cognizant on their process of conducting Batch Implementations.

https://aws.amazon.com/blogs/apn/cognizant-optimized-approach-to-modernize-batch-implementations-on-aws/

Enterprise customers are modernizing existing workloads to cloud for various reasons.

Cost optimization has always been a conversation starter, but that leads to application modernization and operational excellence, which brings agility to application development and enables faster **go-to market (GTM).**

The evolution of the cloud service providers has helped modern applications achieve fluctuating and high demands of infrastructure, software-defined networking and storage, and seamless integration between various managed services.

Modernize Batch Implementation

Bridging the gap between the infrastructure-as-a-service (IaaS) and integration-platform-as-a-service (IPaaS) platform models for modernizing batch processing applications on AWS.

This can lead to 60% performance efficiency and 40% operational cost reduction by leaving the IPaaS provider's complex license model behind.

Cognizant's customer is one of the world's most-used business credit report providers, based in Europe and serving 365 million businesses worldwide.

Before this modernization effort, data pertaining to companies and business institutions from various countries and fetched from different data sources are processed and loaded into the customer's database, and made available to various downstream systems.

Modernize Batch Implementation

Existing Architecture

The customer's existing batch processing application implementation and delivery were accomplished using an integration software from the IPaaS provider and was hosted on their platform.

The source data from different countries were loaded into on-premises database tables. For every country, there was a different schema available and new data was available in the relevant tables on a daily, weekly, or monthly basis depending on the country.

Cron schedulers were used to initiate the batch process for each country on a daily basis. Source data for a country, if available, was extracted from the country-specific tables and loaded into the staging tables. After extraction, sanitized data from the staging tables was inserted into different target tables.

Modernize Batch Implementation

Existing Architecture

Broadly, the batch process consists of following activities:

Cron scheduler: Cron scheduler initiates the batch processes for the various countries on given schedules.
Cleanup stage: Existing data in the staging tables are deleted before any new data is inserted.
Extract stage: Source data is fetched from different on-premises source databases.
Load data stage: Extracted data is loaded to different staging tables
Load translated data stage: This is an optional stage, where data in foreign (non-English) languages are translated before further processing.
Delete duplicate data stage: Duplicate records in the staging tables are deleted.
Load target data stage: In this final stage, sanitized data from the staging tables are loaded into the target tables.

Modernize Batch Implementation

Implementation Scope

Integration platforms typically support a set of components and constructs to define and specify integrations flows using a domain-specific language (DSL) that is unique to the platform.

The applications in our modernization scope were no exceptions:

- Handle multiple extract, transform, load (ETL) batch jobs running concurrently, but at the same time ensure that if a given entity's batch process is already running then the system shouldn't start another batch process for the same entity.
- For individual jobs running, we have to capture the job-level statistics.
- Handle approximately 50 million records for every batch job.
- Enabling a new entity's batch process sould be seamless and not require a new production deployment.
- Create a generic and reusable notification mechanism to the customer's dedicated communication channel.
- Handle compute environment-related failures and develop a notification mechanism to notify the failures.

Modernize Batch Implementation

Implementation Scope

The solution required business information from 30 different countries (regions) to be consumed as a part of an ETL process. The solution also had to handle several new features:

- Ensuring batch processing for a given entity (country or region) does not overlap at any given time.
- Provide job-level custom metrics.
- Resume processing of failed jobs from the point-of-failure of a batch process.
- Handle approximately 50 million records of data for each entity.
- Also, keeping the architecture configuration-driven means that to enable/disable any new batch process requires zero code changes or deployment. Only configuration changes and uploading specific SQL query files would suffice to integrate with this system.

With a wide array of managed and infrastructure services to choose from for compute, network, and storage, designing an elastic, highly available, cost-effective, and scalable solution architecture with all the right components in place was a critical success factor for the modernization project.

Modernize Batch Implementation

Solution Architecture

The modernized solution uses AWS Lambda and AWS Batch Compute on AWS Fargate, a fully managed serverless compute service with workload-aware cluster scaling logic, maintaining event integrations and managing runtimes.

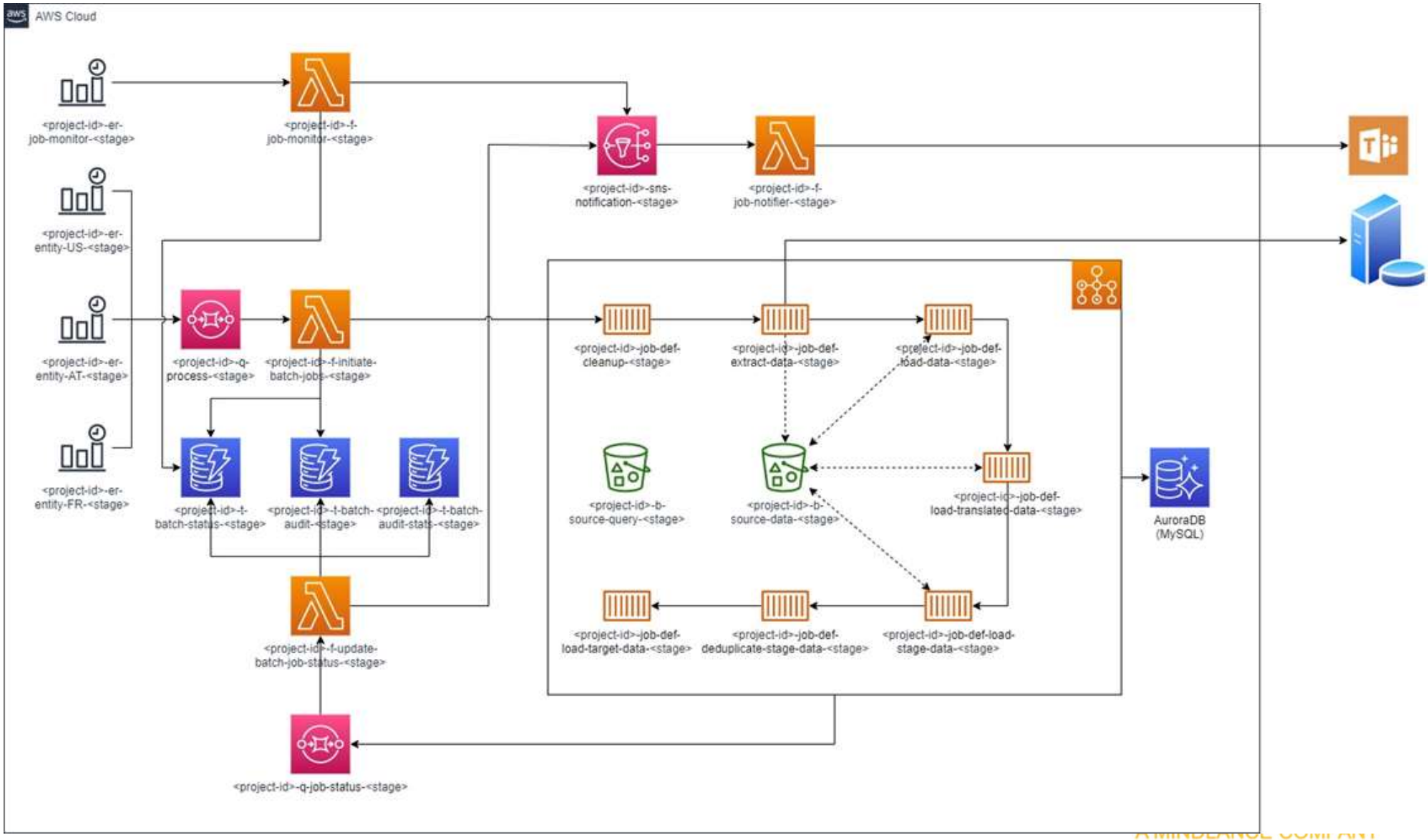
Created compute jobs running as Docker containers and handled the orchestration layer using a serverless architecture.

The solution depends heavily on the availability and reliability of configuration documents for each of the entity's batch processes. It leverages Amazon Simple Storage Service (Amazon S3), a fully managed object storage service that offers industry-leading scalability, availability, security and performance.

It also leverages the Amazon DynamoDB database, a fully managed, multi-region, multi-active, durable NoSQL database solution.

Modernize Batch Implementation

Solution Architecture



Modernize Batch Implementation

The following AWS components were used in the architecture, and here is the context in which they have been used in the solution:

Amazon DynamoDB is used as an idempotent store to track execution status of different entity's batch jobs, as well as to maintain the batch jobs audit records and job statistics data.
AWS Lambda is used to trigger AWS Batch jobs submission to job queue according to defined job dependencies. It checks Amazon DynamoDB for the current batch jobs execution status and then submits the jobs accordingly.
AWS Batch is used to dynamically provision the optimal quantity and type of compute resources based on volume and specific resource requirements of the batch jobs submitted.
Amazon Simple Queue Service (SQS) helped to decouple and scale distributed systems and serverless applications. It's used to de-couple the different AWS service integrations and allow the messaging process between different components.
Amazon Simple Notification Service (SNS) is used for sending notification messages to Teams channel via Lambda integration.
Amazon RDS for MySQL is used as the target database system for the data loading jobs.
Amazon CloudWatch is the primary monitoring solution for applications and infrastructure, providing operational data in the form of logs, events, and metrics. The different entity-specific cron-expressions are configured as CloudWatch Event Rules to trigger the controller layer to initiate the batch jobs for given entity.
Amazon S3 is used to store the different entity-specific SQL query files, thereby externalizing the queries. It's also used to store the extracted data as compressed CSV zip file, which is then consumed by various other batch jobs.

Survey IVR system with Amazon Connect

Building a survey IVR system with Amazon Connect
by Naveen Narayan
05 AUG 2019
Amazon Connect, Contact Center

https://aws.amazon.com/blogs/contact-center/building-a-survey-ivr-system-with-amazon-connect/

Contact centers commonly employ phone-based survey applications. These applications use interactive voice response (IVR) to obtain feedback on agent performance, the ease of conducting a transaction, or both. This post describes how to build a survey IVR using Amazon Connect. Although typical survey accept rates can vary depending on your workflow, coding a survey application using legacy systems often requires a full software development cycle.

Survey IVR system with Amazon Connect

Prerequisites

To follow along with the solution presented in this post, you need the following AWS services and features:

- AWS Lambda
- Amazon DynamoDB
- Amazon Connect
- Amazon Kinesis Data Streams
- Amazon CloudWatch
- AWS Identity and Access Management (IAM)
- You also need an active AWS account with the permission to create and modify Lambda functions and DynamoDB tables.

To begin, you need an Amazon Connect instance configured for inbound and outbound calls. Claim a phone number after you create your instance.

Survey IVR system with Amazon Connect

Solution overview

Contact centers commonly use two types of surveys:

1. In-call survey: The customer opts into the survey in the IVR, then speaks to an agent and ends the call. The system calls the customer back for the survey.
2. External survey: The customer opts into the survey while talking with the agent. After the customer hangs up with the agent, the system calls the customer back for the survey.

The agents' interaction with the customer may be biased during an external survey, as they know whether the interaction may be used to gauge their performance. An in-call survey is less biased as the agent remains unaware if the customer opted into the survey. This blog post presents an option to implement an in-call survey.

Survey IVR system with Amazon Connect

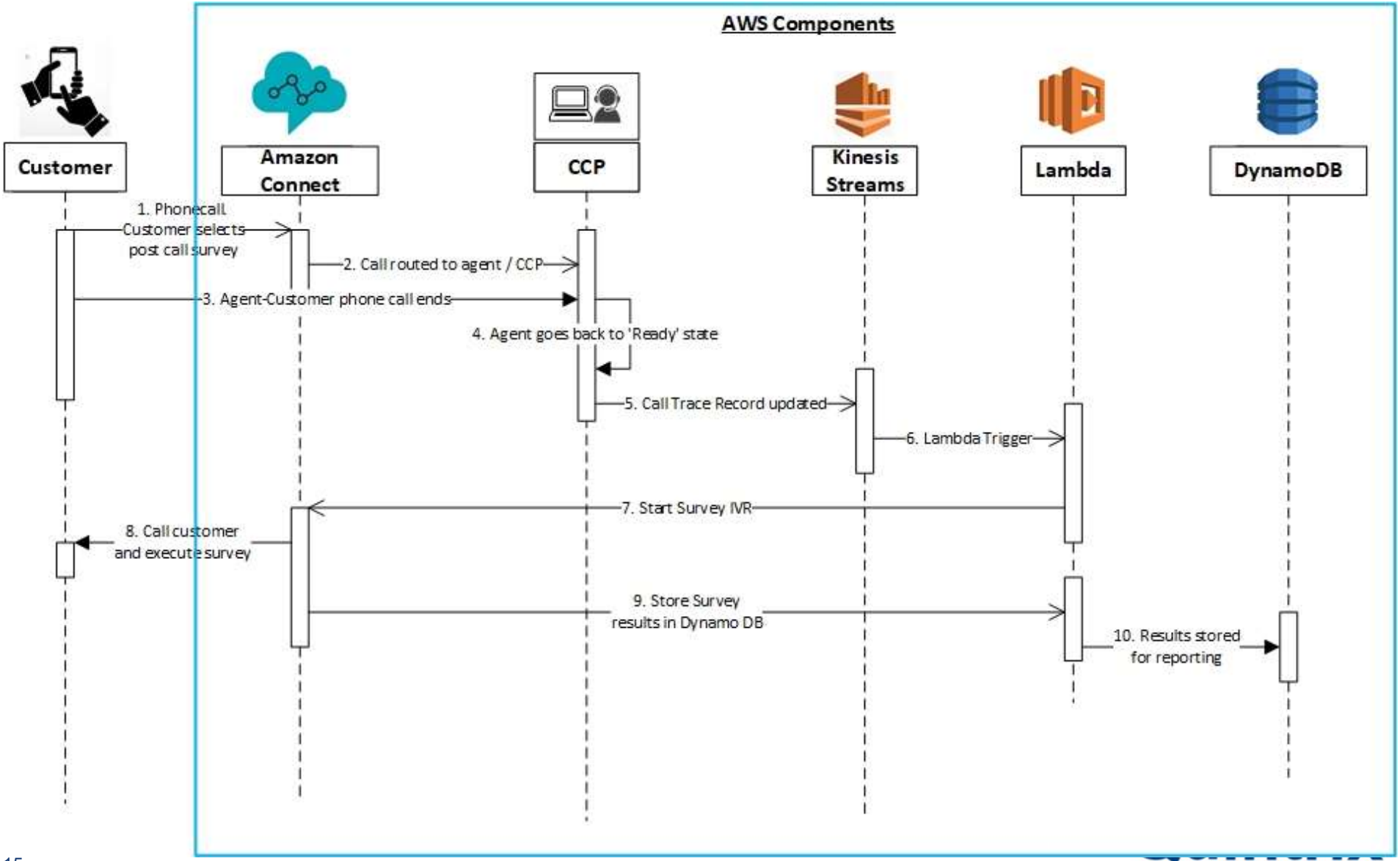
Solution overview

The survey sequence consists of the following steps:

1. The customer calls into the IVR. The IVR offers a survey option after the call. The customer opts in for a survey, and the system transfers the call to an agent.
2. The call routes to the most available agent via their Contact Control Panel (CCP).
3. The customer and agent talk. The customer ends the call.
4. Amazon Connect automatically moves the agent to the “after contact work” state. The agent manually changes to the “available” state.
5. Amazon Connect updates the contact trace record (CTR). This setup requires configuring CTR streaming via Amazon Kinesis Data Streams.
6. Once Amazon Connect finishes updating the CTR, the Kinesis data stream triggers an AWS Lambda function.
7. The Lambda function starts the survey IVR.
8. The survey IVR calls the customer and presents the survey question.
9. The survey IVR stores the customer’s survey response in a DynamoDB table for future analytics.

Survey IVR system with Amazon Connect

Solution overview



Survey IVR system with Amazon Connect

Walkthrough

To build the survey IVR, this post includes the following steps:

- Preparing the environment
- Setting up DynamoDB tables
- Setting up and configuring Kinesis Data Streams in Amazon Connect
- Setting up the Lambda functions
- Logging in to the CCP
- Configuring the Amazon Connect contact flow

Survey IVR system with Amazon Connect

Preparing the environment

You must implement several IAM policies to grant Lambda access to prepare the environment. The Lambda handler requires the following identity-based IAM policies.

Lambda #1 (ProcessCCPEventStream): Kinesis Data Streams triggers these policies when a CTR is ready.

Kinesis read-only access: [CF1] The policy name used for this is = AmazonKinesisReadOnly
[CF1]Please include a sentence explaining what this policy does.

Survey IVR system with Amazon Connect

Preparing the environment

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:Get*",
                "kinesis:List*",
                "kinesis:Describe*"
            ],
            "Resource": "*"
        }
    ]
}
```

Survey IVR system with Amazon Connect

DynamoDB read/write access to the SurveyCustomerInfo table: This policy grants read/write access to the customer survey table. To create a policy that has access to read and write to a specific table, see Amazon DynamoDB: Allows Access to a Specific Table. Replace “MyTable with SurveyCustomerInfo. Policy Name = DDBSurveyCustInfo

Survey IVR system with Amazon Connect

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ListAndDescribe",
         "Effect": "Allow",
         "Action": [
            "dynamodb:List*",
            "dynamodb:DescribeReservedCapacity*",
            "dynamodb:DescribeLimits",
            "dynamodb:DescribeTimeToLive"
         ],
         "Resource": "*"
      },
      {
         "Sid": "SpecificTable",
         "Effect": "Allow",
         "Action": [
            "dynamodb:BatchGet*",
            "dynamodb:DescribeStream",
            "dynamodb:DescribeTable",
            "dynamodb:Get*",
            "dynamodb:Query",
            "dynamodb:Scan",
            "dynamodb:BatchWrite*",
            "dynamodb:CreateTable",
            "dynamodb:Delete*",
            "dynamodb:Update*",
            "dynamodb:PutItem"
         ],
         "Resource": "arn:aws:dynamodb:*:*:table/ SurveyCustomerInfo"
      }
   ]
}
```


Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

Amazon Connect: This policy places the outbound call. Policy Name = AWSConnect-GrantOutboundPermission.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "connect:StartOutboundVoiceContact",
         "Resource": "arn:aws:connect::*:instance/<InsertAmazonConnectInstanceName>/contact/*"
      }
   ]
}
```

NOTE: Change <InsertAmazonConnectInstanceName> with your Amazon Connect instance name.

Write to CloudWatch Logs. Policy name = WriteCloudWatchLogs. JSON noted below.

```
{
"Version": "2012-10-17",
"Statement": [{
         "Action": ["logs:*"],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
 }
```

Survey IVR system with Amazon Connect

Lambda #2 (UpdateSurveyResult): This handler writes the survey results to the DynamoDB table labeled SurveyCustomerInfo. This is the same policy created for the previous DynamoDB read/write access to the SurveyCustomerInfo table policy.

Setting up the DynamoDB table

Create the table SurveyCustomerInfo. This table serves as a lookup table to retrieve the customer's name. It also stores survey results. Configure the table with the following attributes:

- Attribute: 'PhoneNumber'
- Key Type: Primary Key/Partition Key
- Attribute Type: String
- Table Region: US-east-1.

Survey IVR system with Amazon Connect

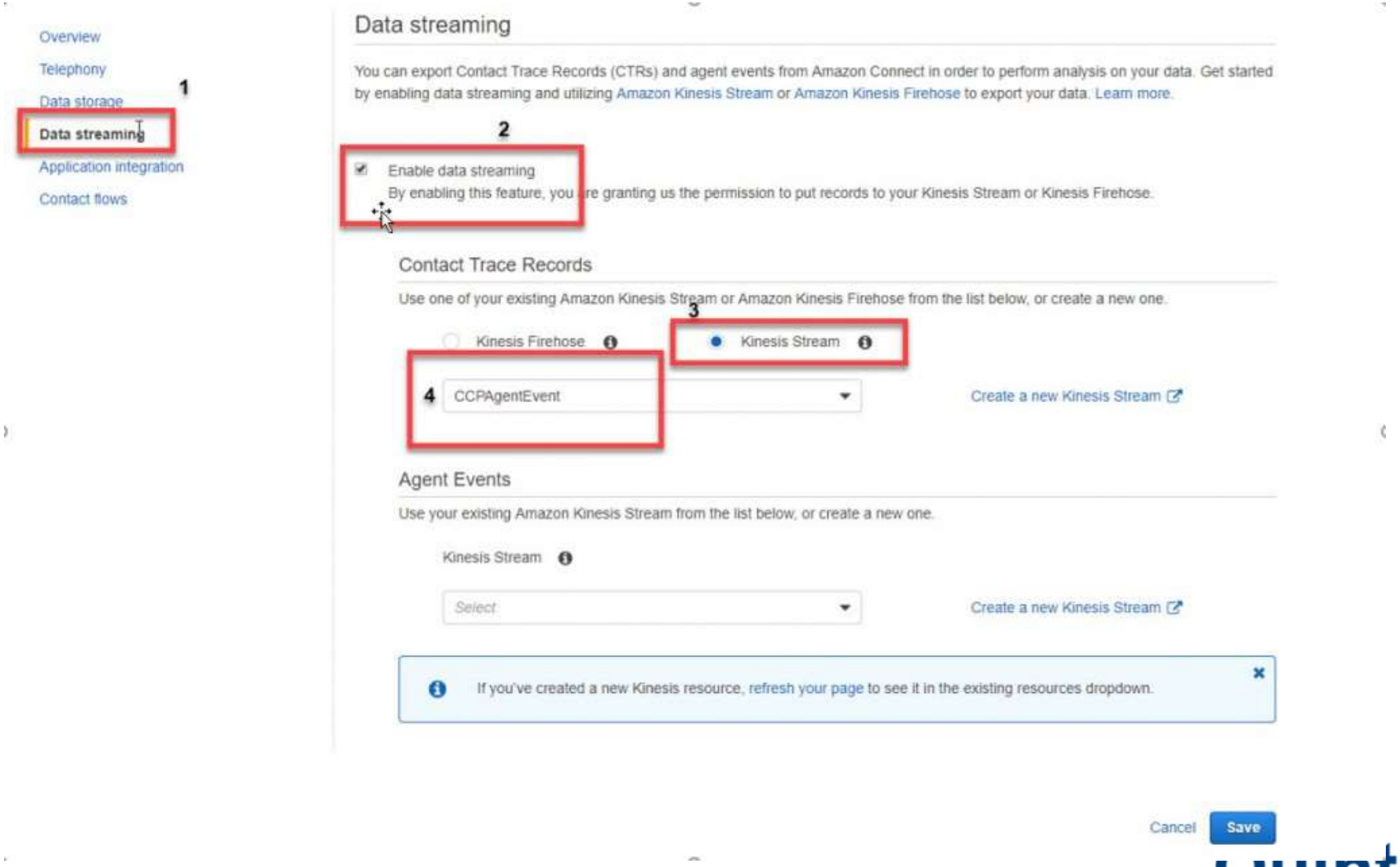
Setting up Amazon Connect to send CTRs to Kinesis Data Streams

Export CTRs via Kinesis Data Streams so that the Lambda function outlined in the next section can read it. After you set up your Kinesis data stream—named CCPAgentEvent in this example—navigate to the Amazon Connect Instance landing page. From here, following these steps to configure Amazon Connect to use that stream:

1. Navigate to the Amazon Connect instance setup screen. Choose Data streaming.
2. Choose Enable data streaming.
3. Under Contact Trace Records, select Kinesis Stream.
4. Type the name of your Kinesis data stream in the text box (CCPAgentEvent in this example).

Survey IVR system with Amazon Connect

Setting up Amazon Connect to send CTRs to Kinesis Data Streams



Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

Setting up the Lambda functions

This survey IVR system requires two new Lambda functions:

1. ProcessCCPEventStream
2. UpdateSurveyResult

After you set up the Kinesis data stream CCPAgentEvent, add it as a trigger to this Lambda function.

Lambda function: ProcessCCPEventStream

This Lambda function requires the following IAM policies:

1. WriteCloudWatchLogs
2. DDBSurveyCustInfo
3. AWSConnect-GrantOutboundPermission
4. AmazonKinesisReadOnlyAccess

Survey IVR system with Amazon Connect

Python

```
import json import base64 import boto3 from boto3 import resource from boto3.dynamodb.conditions import
Key dynamodb_resource = resource('dynamodb') surveyCustomerTable =
dynamodb_resource.Table('SurveyCustomerInfo') CCPEventSource = 'arn:aws:kinesis:us-east-
1:NNNNNNNNN:stream/CCPAgentEvent' ############################################################ ##
Connect Instance Details - edit to change ## Contact Flow Name = OutboundLocationBasedCallback
############################################################ theAmazonConnectPhoneNumber =
'+19729999999' theInstanceId='f633f7ae-84ad-4755-9fca-NNNNNNNNNNN' theContactFlowID = '351c39fb-273a-
4570-a9bc-NNNNNNNNNNN' def lambda_handler(event, context): # print('lambda_handler(event) ==>', event)
for aRecord in event['Records'] : eventSource = aRecord['eventSourceARN'] if eventSource ==
CCPEventSource : payload = base64.b64decode(aRecord['kinesis']['data']) try: processEvent(payload) except
Exception as ex: return { 'isBase64Encoded': 'False', 'statusCode': 999, 'Echo': 'Exception' } return {
'isBase64Encoded': 'False', 'statusCode': 200, 'Echo': 0 } def processEvent(payload) : jsonPayload =
json.loads(payload.decode('utf-8')) isSurveyCandidate = '' print('processEvent(payload) ==> ',
jsonPayload) try: isSurveyCandidate = jsonPayload['Attributes']['SurveyCandidate'] except Exception as
ex: raise ex if isSurveyCandidate == 'True' : phoneNumber = jsonPayload['CustomerEndpoint']['Address']
print('Will call ', phoneNumber,' for survey') invokeOutboundIVR(phoneNumber, jsonPayload['ContactId'])
def invokeOutboundIVR(customerPhoneNumber, contactID) : theConnecClient = boto3.client('connect') try:
response = theConnecClient.start_outbound_voice_contact( DestinationPhoneNumber= customerPhoneNumber
,ContactFlowId=theContactFlowID ,InstanceId=theInstanceId ,SourcePhoneNumber=theAmazonConnectPhoneNumber
,Attributes={'OriginalContactID': contactID} ) except Exception as ex: raise ex def
retrieveCustomerName(aPhoneNumber): try: response = surveyCustomerTable.query(
KeyConditionExpression=Key('PhoneNumber').eq(aPhoneNumber) ,ScanIndexForward=False ,Limit=1 ) except
Exception as ex: raise ex else : if 1 == response['Count'] : return response['Items'][0]['CustomerName']
else : return None
```


Quintrix
A MINDLANCE COMPANY

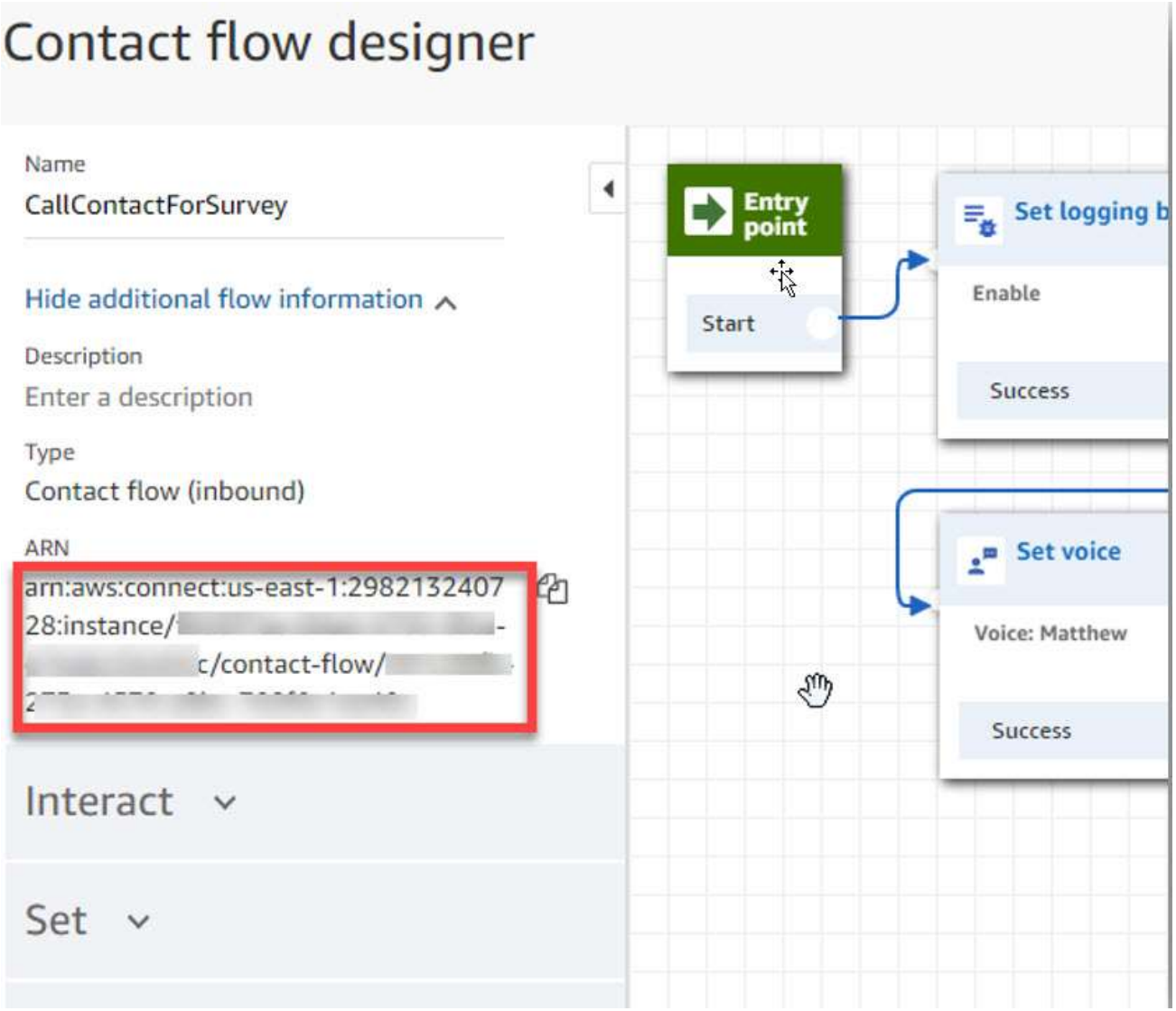
Survey IVR system with Amazon Connect

Update the following four fields after creating the Amazon Connect outbound survey call flow:

1. CCPEventSource: This is the Amazon record locator (ARN) for the data stream.
2. theAmazonConnectPhoneNumber: This is the phone number selected for the Amazon Connect call flow BasicCallFlow described later in the “Configuring the Amazon Connect Contact Flow” section.
3. theInstanceId: This is the instance ID of the Amazon Connect instance.
4. theContactFlowID: This is the contact flow ID of the OutboundSurvey contact flow (described below).

You can find both the instance ID and the contact flow ID listed in the ARN within the Additional flow information panel of the Contact flow designer.

Survey IVR system with Amazon Connect



Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

Function #2: UpdateSurveyResult

This Lambda function requires the WriteCloudWatchLogs and DDBSurveyCustInfo IAM policies

Python

```
import json
import boto3
import datetime
import time
from boto3 import resource
from boto3.dynamodb.conditions import Key

dynamodb_resource   = resource('dynamodb')
surveyCustomerTable = dynamodb_resource.Table('BlogTestSurveyCustomerInfo')

def lambda_handler(event, context):
    print('def lambda_handler(event, context): ==> ', event)
    aContactID  = event['Details']['ContactData']['Attributes']['OriginalContactID']
    aRating     = event['Details']['ContactData']['Attributes']['SurveyResponse']
    aPhoneNumber= event['Details']['ContactData']['CustomerEndpoint']['Address']

    updateSurveyResults(aContactID, aPhoneNumber, aRating)
    return {
        'statusCode': 200,
        'body': 'Ok'
    }


def queryPhoneNumber(aPhoneNumber):
    try:
        if aPhoneNumber:
            response = surveyCustomerTable.query(KeyConditionExpression=Key('PhoneNumber').eq(aPhoneNumber))
    except Exception as ex:
        return None
    else :
        return response


def updateSurveyResults(aContactID, aPhoneNumber, nRating) :
    if aPhoneNumber is not None and nRating is not None :
        surveyData = returnSurveyDataObject(aContactID, nRating)

        retVal = queryPhoneNumber(aPhoneNumber)
        if retVal['Count'] == 0 :
            response = surveyCustomerTable.put_item(Item={
                    'PhoneNumber': aPhoneNumber,
                    'SurveyData': [surveyData]
                })
        else :
            try:
                response = surveyCustomerTable.update_item(
                        Key={
                          'PhoneNumber': aPhoneNumber
                          },
                        UpdateExpression="set SurveyData = list_append(SurveyData, :obj)",
                        ExpressionAttributeValues={
                        ':obj': [surveyData],
                        },
                        ReturnValues="UPDATED_NEW"
```


Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

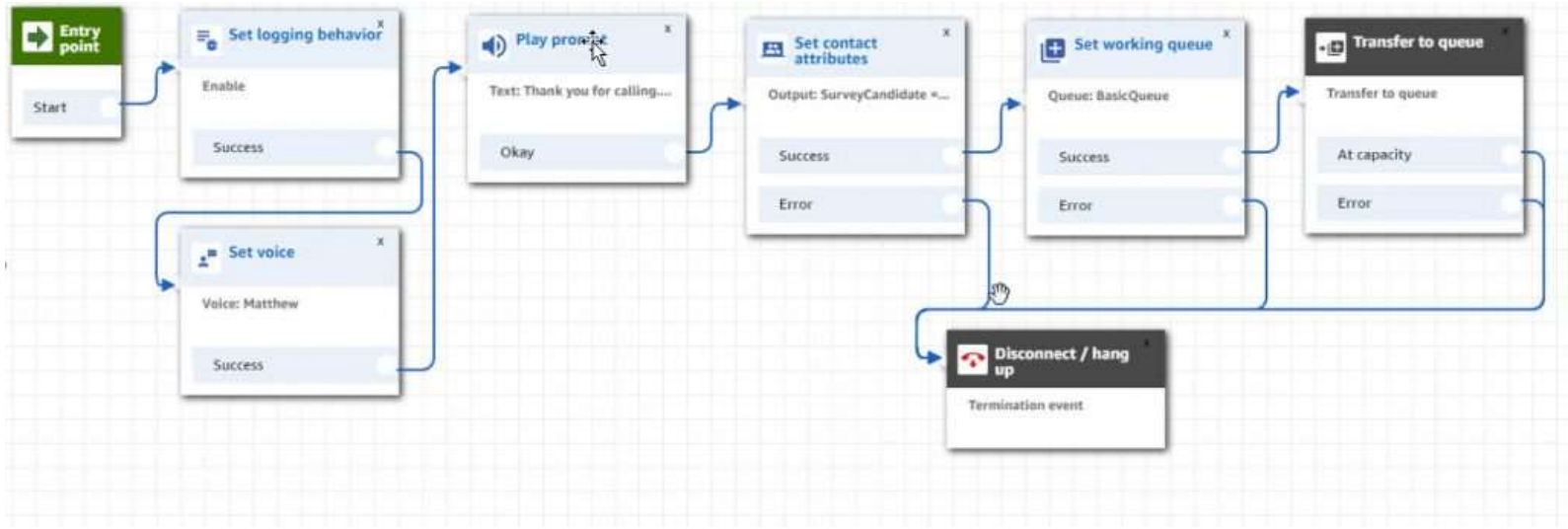
Logging in to the CCP
After setting up your environment, creating your DynamoDB table, and completing your Lambda functions, start and log in to CCP using Amazon Connect.

Configuring the Amazon Connect contact flow
You must configure your Amazon Connect instance to use the Lambda function that you created earlier.

Basic call flow to route to an agent (Flow Name = BasicCallFlow): The basic call flow to send a call to an agent follows a simple process.

- Set up the Play prompt block to speak the following text: “Thank you for calling. We will call you for a one question survey at the end of the call.”
- Set up the Set contact attributes block as follows:
- Use Text / Destination Key = SurveyCandidate / Value = True
- Set up the Set working queue block to pick the Basic Queue. Make sure to map the agent’s routing profile to this queue.
- Make sure that you select and assign a phone number to the call flow after publishing and saving it.

Survey IVR system with Amazon Connect



Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

Survey IVR call flow (Flow Name = OutboundSurvey): The basic call flow to survey a customer follows a similarly simple process.

1. Set up the Store customer input box to play the following text and capture the customer's response. "How likely are you to recommend us to a friend on a scale of 1 to 5. 1 being the least likely and 5 being most likely."
2. Set up the Set contact attributes box to store values entered by the customer in a local variable called SurveyResponse. Use this configuration:
3. Use Attribute : Destination Key = SurveyResponse. Type = System. Attribute = Stored Customer Input
4. Set up the Play prompt block as output from the Set contact attributes block. Have it play the following: "Thank you for rating us $.Attributes.SurveyResponse."
5. The variable $.Attributes.SurveyResponse allows playback of the rating selected by the customer.
6. Set up the Invoke AWS Lambda function box to store the attributes Survey Results in a DynamoDB table by calling the Lambda function UpdateSurveyResult.

Survey IVR system with Amazon Connect

Execution

If you followed the steps to build your survey IVR system, it fulfills the following steps.

1. Call the phone number assigned to the flow BasicCallFlow. The flow automatically enters the caller for a callback survey. (this example uses the caller ID +12549999999)
2. After the agent interaction completes and the agent sets themselves to “Available,” the Lambda Function ProcessCCPEventStream invokes the outbound call flow OutboundSurvey. This flow uses the same number assigned to the BasicCallFlow and calls the customer on their phone number (+12549999999) with a single survey question.
3. After the customer responds to the survey question, the system stores the response in DyanmoDB along with the ContactID for the call and timestamp, as shown in the following JSON:

Survey IVR system with Amazon Connect

Execution

JSON:


```
{ "PhoneNumber": "+12549999999", "SurveyData": [ "{"ContactID": "c84b846e9d4b", "Date": "2019-04-24 17:36:31", "NPS_Rating": "5"}", "{"ContactID":
"dfe518babceb", "Date": "2019-04-24 17:40:03", "NPS_Rating": "2"}", "{"ContactID": "739634da08c0", "Date": "2019-04-24 17:42:37", "NPS_Rating": "4"}",
"{"ContactID": "d1b8fd7211fa", "Date": "2019-05-29 17:14:46", "NPS_Rating": "2"}", "{"ContactID": "99e242603dca", "Date": "2019-05-29 17:16:44",
"NPS_Rating": "1"}", "{"ContactID": "3d1b47b81a23", "Date": "2019-05-29 17:22:03", "NPS_Rating": "1"}", "{"ContactID": "75a8eaa33b9c", "Date": "2019-05-
29 17:24:04", "NPS_Rating": "2"}", "{"ContactID": "b07007156e49", "Date": "2019-05-29 17:25:56", "NPS_Rating": "1"}", "{"ContactID": "365a2c17cf99",
"Date": "2019-05-30 18:07:09", "NPS_Rating": "2"}", "{"ContactID": "077f5577d09a", "Date": "2019-05-30 19:18:50", "NPS_Rating": "2"}", "{"ContactID":
"7ef8ff792000", "Date": "2019-05-30 19:55:37", "NPS_Rating": "2"}" ] }
```


Quintrix
A MINDLANCE COMPANY

Survey IVR system with Amazon Connect

Conclusion

In this post, you created a basic survey application using Amazon Connect. You can now extend this concept to create complex dynamic surveys, where the questions change based on the IVR transactions. Results can be used to provide insight into your IVR application.

The example in this post uses DynamoDB to store customer information. In actual scenarios, you could substitute a CRM database, either on-premises or in the cloud. In this case, you must modify the calls from Lambda.

This post uses a simple CCP. In actual scenarios, you can place the check box within a larger CRM application.

Although this post required the agent to mark a call for a survey, you can eliminate this step and have the customer opt in for the survey in the IVR itself. In this case, the Lambda function to call Amazon Connect and conduct a survey should call only if the customer first opted to participate in the survey.