Contexto

En el proyecto en el que estoy trabajando actualmente (AiVORA) ofrece un asistente virtual multicanal para empresas del mundo auto (Concesionarios, Talleres, Fabricantes, etc).

El producto nació originalmente para ayudar a los agentes de call center con las llamadas que no podían atender, generando tickets con la información recopilada por el asistente durante la conversación. Más adelante se incorporaron canales escritos como WhatsApp o Telegram.

Actualmente estamos en fase de construcción de un portal web donde los agentes del call center pueden entrar a gestionar los tickets generados por el asistente, responder a los clientes mediante llamada o chat y ver información detallada de cada interacción (escuchar llamadas previas, ver chats, detectar peticiones duplicadas, etc)

Otro tipo de usuarios de la plataforma serán los directores y manager, que necesitan ver información agregada sobre el performance del asistente y de los miembros del call center.

Actualmente la información que guardamos de las interacciones con el asistente son el audio de la llamada y la transcripción además de las sesiones de chats. (Alojadas en Firestore)

Diseño del DAaaS

### Definición la estrategia del DAaaS

*Crear una sección de “next best action” para los agentes en la plataforma, de modo que durante la conversación con un cliente obtengan recomendaciones sobre productos que pueden ofrecer o ayuda para resolver la solicitud del cliente*

### Arquitectura DAaaS

* VM como servidor para alojar el portal web
* API Gateway + Google Cloud Functions para la api
* PostgreSQL como base de datos de recomendaciones y para el resto de tablas necesarias para el funcionamiento del portal web
* Firestore base de datos de conversaciones.
* Cloud Storage para almacenar los archivos de conversaciones y recomendaciones
* Cloud Function + Cloud Scheduler para mover las conversaciones desde firebase al un bucket de Cloud Storage una vez al día
* Cloud Function + Cloud Scheduler para arrancar el Dataproc una vez al día
* Dataproc para ejecutar el proceso de generación de recomendaciones
* Cloud Functions + Pub/Sub para insertar el archivo de recomendaciones en Cloud SQL desde el bucket de Cloud Storage

### DAaaS Operating Model Design and Rollout

* *Cada vez que el asistente virtual completa una conversación con un cliente, estructura la información que ha recuperado y llama a una Cloud Function que guarda la conversación en Firestore.*
* *Una vez al día durante la noche, Cloud Scheduler llama a una Cloud Function para que extraiga todas las conversaciones del día anterior de Firestore y las guarde en Cloud Storage*
* *Una vez al día, un lingüista computacional analiza las conversaciones cargadas en Cloud Storage y las etiqueta en base a las directrices comerciales de la empresa. Genera un nuevo fichero de conversaciones etiquetadas*
* *Cada vez que la empresa necesita cambiar las directrices comerciales el administrador del sistema carga estos archivos en Cloud Storage*
* *Una vez al día durante la noche, Cloud Scheduler llama a una Cloud Function para que arranque un Dataproc. Este Dataproc utiliza un script que, tras el arranque de las máquinas, procesa todas las conversaciones etiquetadas del día anterior generando recomendaciones comerciales. Se programa el apagado en la configuración. El archivo generado se guarda en Cloud Storage*
* *Cada vez que se genera un archivo de recomendaciones, se publica un evento en Cloud Pub/Sub. Una Cloud Function se suscribe a este evento y cuando se levanta, ejecuta una carga del archivo de recomendaciones desde Cloud Storage a Cloud SQL*
* *Las recomendaciones son accesibles para los comerciales a través del portal web y la api construida con Cloud Functions y un API Gateway. La api conecta el portal web con la base de datos PostgreSQL en Cloud SQL*

### Desarrollo de la plataforma DAaaS. (ligera descripción del desarrollo)

*Swagger API*

*# openapi2-functions.yaml*

*swagger: "2.0"*

*info:*

*title: api-portal*

*description: Sample API on API Gateway with a Google Cloud Functions backend*

*version: 1.1.1*

*schemes:*

*- https*

*produces:*

*- application/json*

*host: api-portal-v1-1-0-0mwfqqvx.ew.gateway.dev*

*x-google-endpoints:*

*- name: api-portal-v1-1-0-0mwfqqvx.ew.gateway.dev*

*allowCors: True*

*paths:*

*/recomendation/customer/{customerId}:*

*get:*

*summary: get recomendation by customerId*

*operationId: getRecomendation*

*x-google-backend:*

*address: https://europe-southwest1-project.cloudfunctions.net/api-portal/*

*path\_translation: APPEND\_PATH\_TO\_ADDRESS*

*parameters:*

*- in: path*

*name: customerId*

*type: string*

*required: true*

*description: customer ID*

*responses:*

*"200":*

*description: A successful response*

*schema:*

*type: string*

*Cloud Function API*

*const express = require("express");*

*const bodyParser = require("body-parser");*

*const server = express();*

*server.use(bodyParser.urlencoded({ extended: true }));*

*server.use(bodyParser.json());*

*const { Pool } = require('pg')*

*// /recomendation/:recomendationId*

*server.get("/recomendation/:recomendationId", async (req, res) => {*

*console.log("get /recomendation/:recomendationId");*

*console.log("req.params: " + JSON.stringify(req.params));*

*console.log("req.query: " + JSON.stringify(req.query));*

*var sqlBody = {*

*queryString: `select \* FROM recomendation where customerId = $1`,*

*params: [req.params.recomendationId],*

*};*

*var recomendation = await query(sqlBody, params);*

*console.log("recomendationList", recomendation.data);*

*var jsonObj = {*

*recomendationList: recomendation.data,*

*};*

*res.status(200).send(jsonObj);*

*});*

*async function query(sqlBody, params) {*

*const start = Date.now()*

*const duration = Date.now() - start*

*const pool = new Pool({*

*user: env.DB\_USER,*

*host: env.DB\_HOST,*

*database: env.DB\_NAME,*

*password: env.DB\_PASS,*

*config: { max: 1 }*

*})*

*const queryResponse = await pool.query(sqlBody, params)*

*console.log('executed query', { sqlBody, duration, rows: queryResponse.rowCount })*

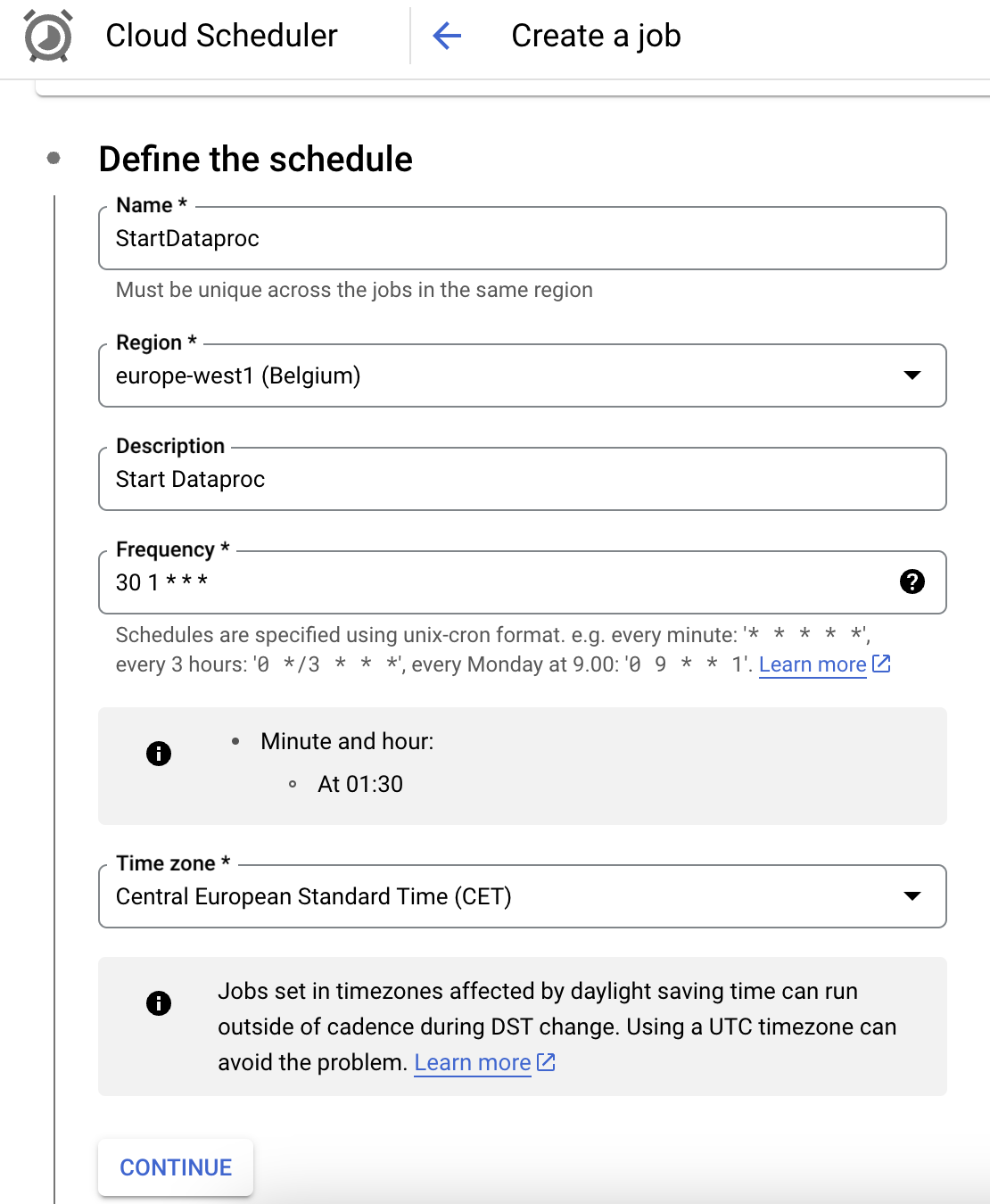
*await pool.end()*

*return queryResponse.rows*

*}*

*exports.httpServer = server;*

Cloud Scheduler



Pub/Sub

gsutil notification create -t recomendationReady -f json -e OBJECT\_FINALIZE gs://recomendations

Cloud Function Dataproc

const dataproc = require('@google-cloud/dataproc').v1;

exports.startWorkflow = (data) => {

const projectId = 'project'

const region = 'europe-southwest1'

const workflowTemplate = 'recomendation-template'

const client = new dataproc.WorkflowTemplateServiceClient({

apiEndpoint: `${region}-dataproc.googleapis.com`,

});

const file = data;

console.log("Event: ", file);

const inputBucketUri = `gs://${file.bucket}/${file.name}`;

const request = {

name: client.projectRegionWorkflowTemplatePath(projectId, region, workflowTemplate),

parameters: {"INPUT\_BUCKET\_URI": inputBucketUri}

};

client.instantiateWorkflowTemplate(request)

.then(responses => {

console.log("Launched Dataproc Workflow:", responses[1]);

})

.catch(err => {

console.error(err);

});

};

Dataproc Template

gcloud dataproc clusters create cluster-a37f --bucket big-data-ai --region europe-southwest1 --master-machine-type n2-standard-4 --master-boot-disk-size 500 --num-workers 2 --worker-machine-type n2-standard-4 --worker-boot-disk-size 500 --image-version 2.1-debian11 --max-age 10800s --initialization-actions 'gs://big-data-ai/Recomendation/recomendation.py' --project big-data-ai

Link a Diagrama:

<https://drive.google.com/file/d/1MqDZYWo1B9VkH1yyn0Syxjj6v7Y5U7Hf/view?usp=sharing>